Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2009

Arturo Miguel Batista Rodrigues

Codificação de vídeo com um único plano de informação

Coding of video with a single information plane

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2009

Arturo Miguel Batista Rodrigues

Codificação de Vídeo com um único plano de informação

Coding of video with a single information plane

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2009

Arturo Miguel Batista Rodrigues

Codificação de Vídeo com um único plano de informação

Coding of video with a single information plane

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor António José Ribeiro Neves, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Armando José Formoso de Pinho, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

Presidente	**Prof. Dr. Tomás António Mendes Oliveira e Silva** Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
Co-Orientador	**Prof. Dr. Armando José Formoso de Pinho** Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
Orientador	**Prof. Dr. António José Ribeiro Neves** Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
Arguente	**Prof. Dr. José Manuel de Castro Torres** Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade Fernando Pessoa

Agradecimentos

Desde logo gostaria de agradecer de forma especial aos meus pais por acreditarem em mim e por terem feito um esforço incansável para me levarem a bom porto a nível académico. Um enorme muito obrigado à minha mãe, por tudo o que me ensinou ao longo destes anos da minha vida. Agradeço também à minha irmã pelo enorme apoio, compreensão e força dada ao longo deste percurso universitário. Sem dúvida, a família é uma pedra basilar no meu sucesso académico.

A todos os meus amigos em Aveiro, que me acompanharam ao longo desta caminhada, com muitos bons momentos de diversão, mas que também estiveram lá nos momentos de maior dificuldade. Tenho de salientar o contributo do meu amigo Ivo Pinheiro, por toda a ajuda dispensada sempre que me surgia alguma dúvida sobre programação!

Endereço também os meus agradecimentos aos meus orientadores, Professor Doutor António José Ribeiro Neves e Professor Doutor Armando José Formoso de Pinho por terem acreditado em mim que poderia levar esta tese a bom porto, assim como pela constante disponibilidade e orientação científica.

Por fim, agradeço também a todos aqueles que possa estar a esquecer-me!

keywords

Vídeo com um único plano de informação, escala de cinzento, vídeos com palete de cores indexada, codificação de Golomb, codificação Aritmética, predição de movimento, reordenação da palete de cores, codificação de vídeo sem perdas.

abstract

As actuais normas para codificação de vídeo, tais como os MPEG2/4 ou H.263/4, foram desenvolvidas para codificação de vídeo com cor. A informação de cor é representada usando um espaço apropriado, como, por exemplo, o YCbCr. Estes espaços de cor são constituídos por três planos: um para a luminância (no exemplo dado, o Y) e dois para a informação de crominância (neste caso, o Cb e o Cr). Contudo, há aplicações onde a informação a codificar é composta apenas por um plano de informação que pode, por exemplo, representar níveis de cinzento em imagem médica, ou índices para tabelas de cores. A motivação desta tese prende-se com dois factos: a produção de imagens médicas em formato digital estar a crescer, impondo técnicas eficazes para o tratamento e a compressão de dados e, embora os modelos de cor indexada sejam há muito utilizados para representar imagens, não têm sido convenientemente explorados em vídeo. Com esta dissertação pretende-se investigar novas estratégias de compressão sem perdas que explorem a redundância entre imagens consecutivas que caracterizam estas modalidades de imagem. Portanto, ao longo do trabalho implementou-se dois codificadores de vídeo para um só plano de informação, baseados num modelo híbrido. Um deles utiliza codificação de Golomb e o outro codificação aritmética, estudando-se assim a eficácia de cada um, quer para a escala de cinzentos, quer para vídeos com tabela de cores indexadas. Adicionalmente, para vídeos de cor indexada, implementou-se um algoritmo de reordenação da tabela de cores, o que torna a codificação mais eficaz.

keywords

Video with a single information plane, gray scale, color indexed videos, Golomb coding, Arithmetic coding, motion prediction, palette reordering, lossless video coding.

abstract

The current standards for video encoding, such as MPEG2/4 or H.263/4, have been developed for encoding video with color. The color information is represented using an appropriate space, such as YCbCr. These color spaces are made of three planes: one for luminance (in the given example, the Y) and two for the chrominance information (in this case, the Cb and Cr). However, there are applications where the information lies in a single information plane that may, for example, represent shades of gray (medical imaging) or indexes to color tables (color indexed video). The motivation of this thesis is related with two points: the production of medical images in digital format has been growing, imposing efficient techniques for the treatment and compression of data and, although color indexed models have been used for a long time to represent images, it has not been adequately explored in video. With this thesis, we intended to investigate new strategies for lossless compression which exploits the redundancy between consecutive images that characterize these types of images. Therefore, during this work, it has been implemented two video encoders with one information plane, based on a hybrid model. One of them uses Golomb codes and the other arithmetic coding. It has been studied the efficiency of each one, both using gray scale and color indexed videos. In addition, for color indexed videos, it has been implemented a palette reordering algorithm, making the encoding more efficient.

Contents

Chapter 1

Introduction

The current standards for video coding have been developed to deal with color video. The color information is represented using an appropriate space, such as YUV or RGB. Both of them are composed by three planes of information: in the case of YUV, Y represents the luminance and U and V the chrominance information; and in the case of RGB, they represent the three additive primary colors: Red, Green and Blue, respectively.

However, sometimes the color information is composed by various shades of gray or a reduced number of colors. On both cases, there is no need to represent the color information in a three plane space, only one is enough. Those are the types of video that we are interested to study: gray scale and color indexed videos.

One example of data in a gray scale format are some data provided in medical diagnosis (an example can be seen in Fig. 1.1). Their production in digital format has been growing and nowadays they are an important and indispensable element of medical decision. With the data increasing, the space to store all those records also increases. Therefore, it is necessary to compress the data to reduce the occupied space but, on the other hand, it is also necessary to maintain the information equal to the original. So, the development of efficient lossless video coding methods is very important. Another example of gray scale videos are the video surveillance tapes that can be found in retail parks, important buildings, malls and even in highways.

Color-indexed images are represented by a matrix of indexes (the index image) and by a color-map or palette (an example is presented in Fig. 1.2). The indexes in the matrix point to positions in the color-map and, therefore, establish the colors of the corresponding pixels. This type of images are obtained by quantizing a full color image to an image with, generally, no more than 256 colors carefully selected. This process is usually considered in two parts:


Figure 1.1: Two examples of medical images in gray scale. The first one is a sonography [1] and the second is a CT Scan [2].

the selection of an optimal color palette and the optimal mapping of each pixel of the image to a color from the palette. The limited ability of humans to differentiate between the full range of representable colors allows the selection of a limited number of colors.

Figure 1.2: Two examples of color indexed images. Both images are the first frame from the video News, found at [3], where the first one has been reduced to a palette of 8 colors, whereas the second one has a colormap of 256.

Even though it is usual to find color-indexed images, such as PNG and GIF images, the same has not been adequately explored in video. As with gray scale videos, encoding this information must be made without loss of information, as the suppression of large quantities of information has been already made with the quantization of the original images.

1.1 Objectives and main contribution

The main objectives of this thesis are the development of several algorithms using different coding methods and then study them in order to assess which one can be more effective, whether in gray scale or in color-indexed videos. We present two video coding methods based on a hybrid approach that use Golomb coding and arithmetic coding.

Moreover, it is important that both types of video can be stored in a common bitstream. In this thesis, it is proposed a bitstream format that contains a common header, which specifies the number of rows, columns, frames per second and the number of colors, followed by the encoded frames. If the encoded video is a color indexed one, all the colormaps are placed right after the corresponding frame.

1.2 Thesis structure

This thesis is divided into six chapters. The second chapter offers an extensive review of the most important theoretical contents about coding, explaining the Golomb and the arithmetic coding, the hybrid approach using motion prediction, the evolution of the MPEG's video coding standards, with special focus in the H.264/MPEG4-AVC standard, the most important lossless image compression standards and, finally, some lossless video coding methods. The third chapter describes the color quantization methods to convert a full color image into a reduced set of colors. It is explained the most important methods of color reduction, inverse colormapping, palette-reordering and dithering. In the fourth chapter, the bitstream and the two developed algorithms are explained in better detail, as well as the conversion of the original source videos into a single plane space, both for gray scale and the color-indexed format. In the fifth chapter, the experimental results with the two developed algorithms are shown, as well as the compression results using the H.264/MPEG4-AVC and the JPEG2000 video and image standards, respectively. Comparisons are made between them in order to see if the developed algorithms are more efficient for encoding gray scale and color-indexed videos using the new proposed model. Finally, in the sixth chapter, some conclusions are made and we present some possible future work.

Chapter 2

Video coding

Nowadays, the constant use of information technologies is unquestionable. The majority of the data are being replaced into a digital format. Even though the storing equipments of digital data are also improving and increasing their capacity, decreasing the size of the data to be stored is still a goal.

Encoding relies on a exploitation of perceptual or statistical redundancy. Perceptual redundancy tries to exploit the limitations of human perceptions, such as the vision and the audition. On the other hand, statistical redundancy occurs when it is possible to define a value with fewer bits than the original. Because this thesis relies on lossless video coding, only the statistical redundancy will be focused.

Statistical methods can use variable length codes, which allow an encoding with zero error and still be read back symbol by symbol. When using these types of codes, the most frequent data symbols will be encoded with less number of bits while less frequent symbols will be represented with longer code words. Another approach usually used is arithmetic coding, which encodes the entire message altogether. Using the right coding strategy, the original data may be compressed close to its entropy. In order to achieve a better efficiency in the compression rate, it is usual to use a hybrid encoder, which mixes spatial coding with motion prediction.

During the elaboration of this thesis it was developed two encoders, one using the Golomb codes and the other the arithmetic coding along with motion prediction, and studied their efficiency when applied in one information plane videos. In this chapter, we describe these concepts. Moreover, it will also be presented the most important video coding methods supporting lossless coding. The description of the standards for lossless coding of images is also presented, due to the fact that they will be used in this thesis to compare the results

with the developed algorithms.

2.1 Golomb codes

The main idea of the Golomb code [4] relies on separating a positive integer into two parts: one of them is represented with an unary code and the other with a binary code. These two parts represent the quotient and the remainder of the initial value to encode. Therefore, the initial value is divided by a previously defined number and the quotient is represented by the unary code and the remainder by the binary code. Let n be an integer greater or equal than 0. It can be represent by two numbers, q and r, in such a way that

$$q = \frac{n}{m} \qquad \text{and} \qquad r = n - q\ m$$

The quotient, q, can have the values 0, 1, 2,..., and the remainder of the division, r, can have the values 0, 1, 2, ...,m-1. The remainder, if written in binary code, is represented by $\lceil \log_2(m) \rceil$ bits.

In order to achieve a better efficiency, it is necessary to choose the best value of m. This number must be chosen according to the entropy of the data, which is given by

$$E = -\sum_{i=0}^{N-1} P(i) log(P(i))$$

where N is the number of possible symbols and $P(i)$ is the probability of each symbol.

According to the entropy value, m should be close to E. If m is not chosen accordingly to the entropy, it must be preferentially a power of 2, otherwise the binary code is not efficient.

2.2 Arithmetic coding

This method [5], opposing to other entropy encoding techniques that separate the input message into its component symbols and replace each symbol with a code word, encodes the entire message into a single number.

Arithmetic coding uses an one-dimensional table of probabilities. The single number to be encoded will be a value inside the probability range. All probabilities fall into the range [0,1) while their sum equals 1 in every case. The [0, 1) interval is partitioned according to the probability distribution of the symbols and then, after iterating this step for each symbol in the message, a value inside the final interval is chosen for representing the message.

To better understand how does the Arithmetic coding work, let's assume the alphabet $A = \{a, b, c, d\}$. Let the probabilities of the symbols in the message be

$$P(a) = 0.5, \qquad P(b) = 0.25, \qquad P(c) = 0.125 \qquad and \qquad P(d) = 0.125.$$

Now the interval [0,1) will be partitioned according to the probability of the symbols, according to Table 2.1.

Symbol	**Prob.**	**Interval**
a	0.5	[0, 0.5)
b	0.25	[0.5, 0.75)
c	0.125	[0.75, 0.875)
d	0.125	[0.875,1)

Table 2.1: The [0, 1) interval partitioned according to the probability distribution of the symbols.

Let S be the message to be encoded. The first step in encoding is the initialization of the interval $I = [low, high)$ by $low = 0$ and $high = 1$. When the first symbol of S is read, the interval I is resized to a new interval according to the symbol. For instance, if the first symbol of S is b, the new boundaries of I are, according to the Table 2.1: $low = 0.5$ and $high = 0.75$. All the following numbers generated by the next iterations will be located in the interval I.

Proceeding with the second symbol of S, it is necessary to scale and shift the boundaries to match the new interval. Scaling is accomplished by a multiplication of the boundaries of the symbol to encode with the difference between *high* and *low*, that is, the length of the interval. Shifting is performed by adding *low* to the result of scaling. Therefore, the equations to find the new boundaries are

$$low' = low + l \times (high - low) \qquad and \qquad high' = low + h \times (high - low),$$

where l and h represent the lower and upper limits of the symbol to encode, respectively.

This rule is valid for all steps, including the first one. Replacing low with 0 and $high - low$ with 1, it is easy to see that the result will be the same as assigning directly the boundaries of the first encoded symbol.

To decode the message, the encoder will be applied backwards. The final value of the encoding algorithm is received by the decoder to restore the original message S. In the first

iteration, it is made a comparison between the encoded value with each interval of the initial partition, to find the one that contains that value. It will correspond to the first symbol of the sequence. To compute the next symbol, the probability partition is modified using the same way as encoding, that is, it will be used the equations referred above.

Although this method is efficient when encoding short messages, the same does not occur with long messages, which may require a infinite precision computer. Even with those computers, it certainly would not be efficient to perform arithmetic operations with several thousands or even millions of decimal places. Therefore, it is necessary to find a way to re-scale the interval.

Analyzing the calculated intervals, it is possible to see a tendency that the limits are getting closer as the new symbols are being encoded. This means that, sooner or later, it will happen one of three situations:

- both limits are under 0.5;
- both limits are above 0.5;
- the limits belong to the interval [0.25, 0.75).

When the first two possibilities happen, looking to the number representation in binary, the most significant bit of the lower and upper limit will always be 0 or 1, if the limits are under or above 0.5, respectively. Therefore, this bit can be sent. Now, the limits will be expanded, according with the following equations:

- $l = 2l$ and $h = 2h$, if the sent bit was 0;
- $l = 2(l - 0.5)$ and $h = 2(h - 0.5)$, if the sent bit was 1.

When the third option happens, in binary it means that the second most significant bit in the lower limit is always 1 and in the upper limit always 0. So, this bit will be deleted in order to re-scale the limits and allow a new calculation of them. This expanding can also be represented by the following equations:

- $l = 2(l - 0.25)$ and $h = 2(h - 0.25)$.

A counter will take note how many times these situations happen between two sent bits and, when one of the other situations happen, after the sending of the corresponding bit to the encoded message, it is also sent a bit to sign that the third situation happened. It will

be sent the 0 bit as many times as the number of the counter, if the sent bit was 1 and vice versa if 0 was the sent one.

There are two ways of applying the one-dimensional table of probabilities, which is also known as finite-context models: statically and dynamically. In this thesis, we used the dynamic method, because using the static method would require the previous knowledge and storage of the data information. Using the dynamic method, the statistical information is adapted along with the encoding process.

A finite-context model [4, 5] of an information source assigns probability estimates to the symbols of an alphabet A, according to a conditioning context computed over a finite and fixed number, M, of past outcomes (order-M finite-context model). The number of conditioning states of the model is $|A|^M$. The probability will be the number of times that, in the past, the value to encode had appeared with that M past outcomes, divided by the number of all the symbols already encoded. The counters are updated each time a symbol is encoded. Since the context template is causal, the decoder is able to reproduce the same probability estimates without needing additional information. At the beginning of the encoding process, the counters of all the possible conditioning states will be initialized with the value 1. In the decoding algorithm, the same method was applied. This method avoids the previous knowledge of the statistics of the message to encode or decode, and can be learned along with the development of the encoding/decoding process.

2.3 Motion prediction

A video sequence is composed by several images, also known as frames. Usually, the difference between two consecutive frames of a video sequence are due to the motion of some elements of the scene (Fig. 2.1). Exceptions occur when there are scene changes, zoom-in or zoom-out operations and camera translation.

Hence, this motion of some elements is another redundancy that can be exploited. The basic steps of video coding based on motion prediction are:

- Estimation of the motion vectors.
- Compensation, i.e., temporal prediction.
- Encoding of the motion vectors.
- Encoding of the prediction residuals.

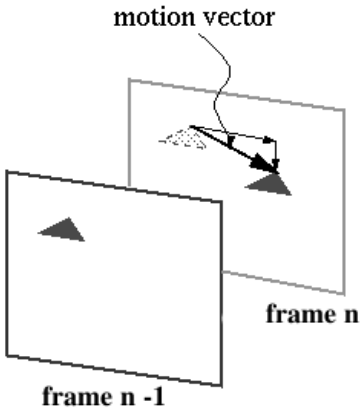

Figure 2.1: Example of motion between two consecutive frames.

There are several techniques where motion prediction is applied. The most common and also the one that was used in this thesis divides the frame in blocks with dimensions chosen previously and, for each block, seeks the position where the difference with the reference block of the previous frame is minimum. For each block, there might be two displacements: one in the horizontal direction and the other in the vertical direction. This forms the motion vector. Usually the search for the best value is limited to a neighborhood of a number of pixels around the block previously chosen, otherwise the computation of this algorithm would be too demanding.

2.4 A little of MPEG's history

MPEG, which stands for Moving Picture Experts Group [6], is the name of a family of standards used for coding audio-visual information in a digital compressed format.

It has started in 1988 as a working group within ISO/IEC with the aim of defining standards for digital compression of audio-visual signals. MPEG's first project, MPEG-1, was published in 1993 as ISO/IEC 11172. It is a three-part standard defining audio and video compression coding methods and a multiplexing system for interleaving audio and video data, so that they can be played back together. MPEG-1 mainly supports video coding up to about 1.5 Mbit/s, giving quality similar to VHS, and stereo audio at 192 bit/s. It is used in the CD-i and Video-CD systems for storing video and audio in CD-ROM.

During 1990, MPEG recognized the need for a second standard for encoding video for broadcast formats at higher data rates. MPEG-2 [7], which is formally known as ISO/IEC-13818, is capable of coding standard-definition television at bit rates from about 3-15 Mbit/s

and high-definition television at 15-30 Mbit/s. MPEG-2 extends the stereo audio capabilities of MPEG-1 to multi-channel surround sound coding. MPEG-2 decoders can also decode MPEG-1 bitstreams. This new standard has been approved in November 1994 and the final text was published in 1995.

A MPEG-3 project was anticipated to be aimed at HDTV, but MPEG-2 was shown to be capable of filling that need and MPEG-3 never occurred.

In October 1998, as a result of an international effort involving hundreds of researchers and engineers from all over the world, it was published as ISO/IEC 14496, the new codec MPEG-4 [8], becoming an International Standard in the first months of 1999. This standard was initially specified for very low bit rates but now it supports much higher bit rates. MPEG-4 is designed for use in broadcast, interactive and conversational environments, allowing to be used in television and Web environments. The second version of this standard, which also includes the entire technical content of the first version of the standard, became an international standard in 2000. After these two versions, more parts were developed, and new tools and profiles were introduced.

An example was the development of Part 10, better known as H.264 or Advanced Video Coding (AVC), which substantially improves MPEG-4's video compression efficiency. This standard will be focused with better detail in the following section.

2.5 H.264

H.264/MPEG4-AVC [9] is a video coding standard developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG), and it became an International Standard in 2003. H.264/MPEG4-AVC has recently become the most widely accepted video coding standard since the deployment of MPEG-2 at the dawn of digital television, and it may soon overtake MPEG-2 in common use. It covers all common video applications ranging from mobile services and videoconferencing to IPTV, HDTV, and HD video storage. This standard is divided into a Video Coding Layer (VCL), whose block diagram is shown in Fig. 2.2, and a Network Abstraction Layer (NAL). NAL's main goal is to provide appropriate headers and system information for each particular network or storage media, which is out of the thesis goal and thus, it will not be explained with more details.

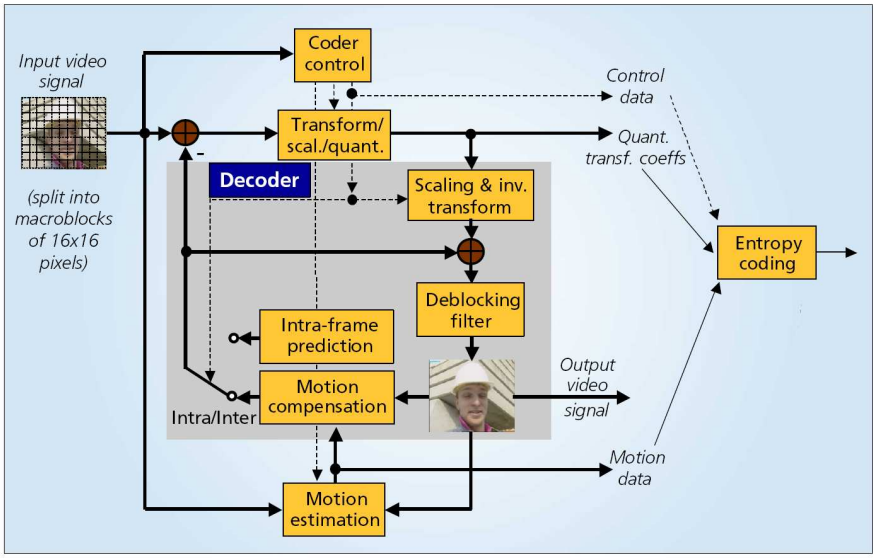

Figure 2.2: VCL block diagram [9].

2.5.1 VCL

The typical encoding operation for a picture begins with splitting the picture into blocks of samples, each one covering a rectangular picture area of 16×16 samples of the luma component and, in the case of video in 4:2:0 chroma sampling format, 8×8 samples of each of the two chroma components. The macroblocks are organized in slices, which represent regions of a given picture that can be decoded independently of each other. In this standard, the allocation of macroblocks into slices is made completely flexible. With this feature, which is informally known as *flexible macroblock ordering* (FMO), a single slice may contain all of the data necessary for decoding a number of macroblocks that are scattered throughout the picture, which can be useful on error recovery due to packet loss. In contrast with previous standards in which the picture type determined the macroblock prediction modes available, it is the slice type that determine which prediction modes are available for the macroblocks. H.264/MPEG4-AVC supports five different slice types. The simplest is the I slice (where I stands for intra). In I slices all macroblocks are coded without referring to any other pictures of the video sequence. Prior coded images can be used to form a prediction signal for macroblocks of the predictive-coded P and B slice types (where P stands for predictive and B stands for bi-predictive). The remaining two slice types are SP (switching P) and SI (switching I) slices, which are specified for efficient switching between bitstreams coded at various bit rates. Slices of different types can be mixed within a single picture.

The first picture of a sequence is typically coded in Intra mode. H.264/AVC uses spatial

directional prediction, in which individual sample values are predicted based on neighboring sample values that have already been decoded and fully reconstructed. Two different modes are supported for the luma block: 4×4 and 16×16 pixels/sample. However, the intra prediction process for chroma blocks uses prediction block size equal to the block size of the entire macroblock's chroma arrays.

In the 4×4 intra coding mode, each 4×4 luma block within a macroblock can use a different prediction mode. Eight different prediction directions plus an additional averaging (so-called DC) prediction mode can be selected by the encoder. Fig. 2.3 and 2.4 illustrate those prediction directions.

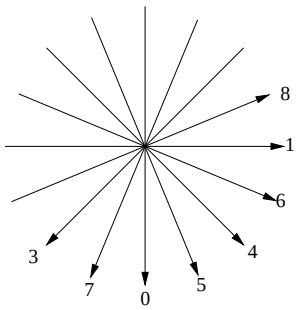

Figure 2.3: Spatial prediction for the 4×4 intra mode [10].

The 16×16 intra prediction mode, depicted in Fig. 2.5, operates similarly to the 4×4 intra mode, except that the entire luma block is predicted at once, based on the samples above and to the left of the macroblock. Also, in this mode there are only four modes available for prediction: DC, horizontal, vertical and planar. This mode is most useful in relatively smooth picture areas.

For all the remaining pictures of a sequence or between random access points, typically Inter coding is used. Inter coding employs inter-picture temporal prediction (motion compensation) using other previously decoded pictures. For P-slice macroblocks, the motion compensation model in H.264/AVC is more powerful and flexible than those defined in earlier standards, which typically allowed motion compensation block sizes of only 16×16 or possibly 8×8 luma samples, whereas this standard allows seven different block sizes: 16×16, 16×8, 8×16, 8×8 (Fig. 2.6), with this last block being further partitioned in 8×8, 8×4, 4×8 and 4×4 samples, as depicted in Fig. 2.7.

The prediction signal for each predictive coded $M \times N$ luma block is obtained by displacing a corresponding area of a previously decoded reference picture, where the displacement is specified by a translational motion vector and a picture reference index. Thus, if the macroblock is coded using four 8×8 sub-macroblocks, and each sub-macroblock is coded using

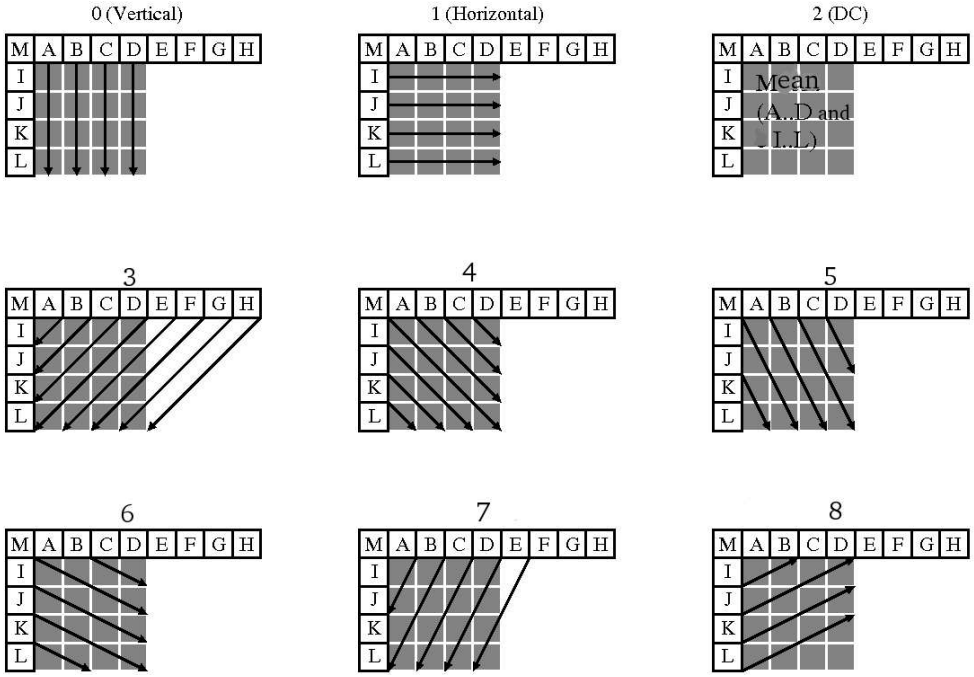

Figure 2.4: The nine prediction types of the 4×4 intra mode, according to the spatial prediction directions shown in Fig. 2.3 [10], adapted from [11].

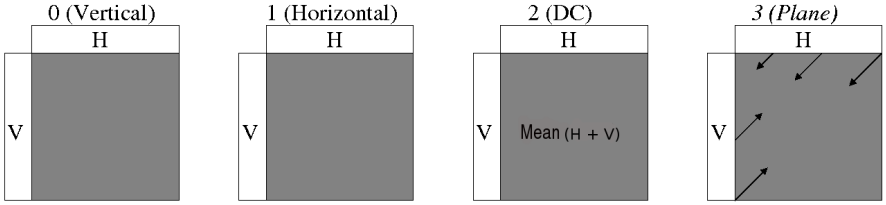

Figure 2.5: Spatial prediction for the 16×16 intra prediction mode [10], adapted from [11].

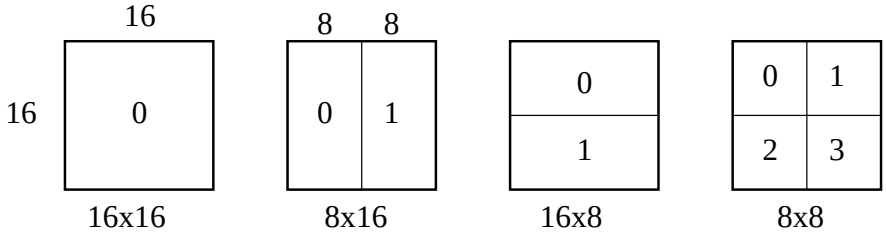

Figure 2.6: Partitioning of a macroblock in 16×16, 16×8, 8×16, 8×8 for motion-compensated prediction [10], adapted from [11].

four 4×4 luma blocks, a maximum of 16 motion vectors may be transmitted for a single *P*-slice macroblock. The motion vector precision is at the granularity of one quarter of the distance between luma samples. If the motion vector points to an integer-sample position,

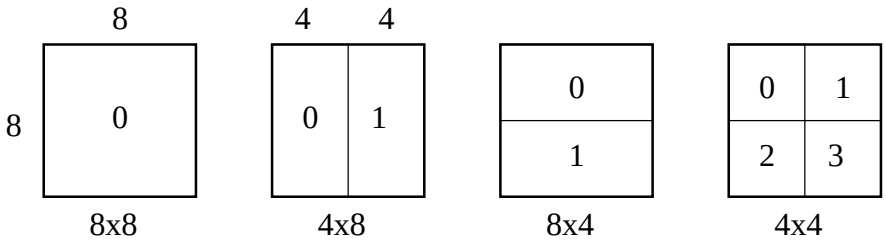

Figure 2.7: Partitioning of the 8×8 macroblock into a 8×8, 8×4, 4×8 or 4×4 sub-macroblock for motion-compensated prediction [10], adapted from [11].

the prediction signal is formed by the corresponding samples of the reference picture. Otherwise, the prediction signal is obtained using interpolation between integer-sample positions. The prediction values at half-sample positions are obtained by separable application of a one-dimensional six-tap finite impulse response (FIR) filter, and prediction values at quarter-sample positions are generated by averaging samples at integer- and half-sample positions. The prediction values for the chroma components are obtained by bilinear interpolation.

H.264/AVC supports multi-picture motion-compensated prediction, which means that, more than one prior-coded picture can be used as a reference for motion-compensated prediction. For multi-frame motion-compensated prediction, the encoder stores decoded reference pictures in a multi-picture buffer. In Fig. 2.8, it is depicted an example of usage of multiple frames to encode the blocks of the current frame.

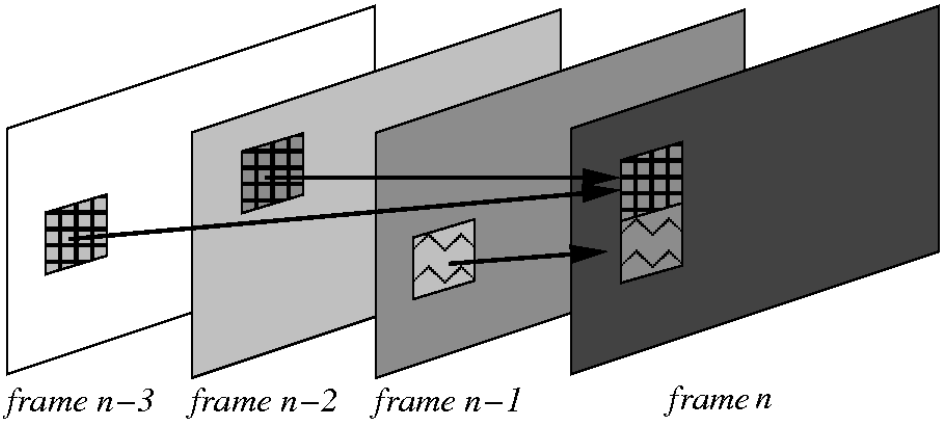

Figure 2.8: Multiframe motion compensation. This concept is applied both to P or B slices [9].

In addition to the motion-compensated macroblock modes described above, a *P*-slice macroblock can also be coded in the so-called SKIP mode. For this mode, neither a quantized prediction error signal, nor a motion vector or reference index parameter, has to be trans-

mitted. The reconstructed signal is computed in a manner similar to the prediction of a macroblock with partition size 16×16 and fixed reference picture index equal to 0. In contrast to previous video coding standards, the motion vector used for reconstructing a skipped macroblock is inferred from motion properties of neighboring macroblocks rather than being inferred as zero.

The main difference between B and P slices is that B slices may use a weighted average of two distinct motion-compensated prediction values for building the prediction signal, to encode the sample value. Motion compensation in B-slices is supported by four different types of inter-picture prediction: list 0, list 1, bi-predictive, and direct prediction. While list 0 prediction indicates that the prediction signal is formed by utilizing motion compensation from a picture of the first reference picture buffer, a picture of the second reference picture buffer is used for building the prediction signal if list 1 prediction is used. In the bi-predictive mode, the prediction signal is formed by a weighted average of a motion-compensated list 0 and list 1 prediction signal. The direct prediction mode is inferred from previously transmitted syntax elements and can be either list 0 or list 1 prediction or bi-predictive.

The residual of the prediction, which is the difference between the original input samples and the predicted samples for the block, is transformed. H.264/MPEG4-AVC specifies a set of integer transforms of different block sizes. A 4×4 integer transform is applied to both the luma and chroma components of the prediction residual signal. An additional $M \times N$ transform is applied to the DC coefficients of each chroma component, with the values of $N, M \in 2, 4$, depending on the format. If a macroblock is coded in Intra-16×16 mode, a similar 4×4 transform is performed for the 4×4 DC coefficients of the luma signal.

The transform coefficients are then scaled and approximated using scalar quantization. This standard uses uniform-reconstruction quantizers (URQs). One of 52 quantizers factors is selected for each macroblock by a quantization parameter (QP). The scaling operations are arranged so that there is a doubling in quantization step size for each increment of six in the value of QP.

The quantized transform coefficients of a block are generally scanned in a zig-zag fashion and further entropy coded and transmitted together with the entropy-coded prediction information for either Intra or Inter-frame prediction. In H.264/AVC, two methods of entropy coding are supported. The default entropy coding method uses a single infinite-extend codeword set for all syntax elements, except the quantized transform coefficients. Thus, instead of designing a different VLC table for each syntax element, only the mapping to the single codeword table is customized according to the data statistics. The single codeword table chosen is an exp-Golomb code with very simple and regular decoding properties. For transmitting

the quantized transform coefficients, a more sophisticated method called Context-Adaptive Variable Length Coding (CAVLC) is employed. In this scheme, VLC tables for various syntax elements are switched, depending on already-transmitted syntax elements. Since the VLC tables are well designed to match the corresponding conditioned statistics, the entropy coding performance is improved in comparison to schemes using just a single VLC table.

The efficiency of entropy coding can be improved further if Context-Adaptive Binary Arithmetic Coding (CABAC) is used. As referred in 2.2, arithmetic coding is extremelly efficient once it permits adaptation to non-stationary symbol statistics. Another important property of CABAC is its context modeling. The statistics of already coded syntax elements are used to estimate the conditional probabilities.

Compared to CAVLC, CABAC can typically provide reductions in bit rate of 10 to 20 percent for the same objective video quality when coding SDTV/HDTV signals [9].

The decoder inverts the entropy coding processes, performs the prediction process as indicated by the encoder using the prediction type information and motion data. It also inverse-scales and inverse-transforms the quantized transform coefficients to form the approximated residual and adds this to the prediction. The result of that addition is then fed into a deblocking filter, which provides the decoded video as its output.

2.6 Lossless image compression standards

In this thesis, we used lossless image coding standards to compare the efficiency of the developed methods or to obtain reference values. Therefore, in this section, it will be described the most common image coding standards available.

2.6.1 The JPEG standard

JPEG [13], which stands for Joint Photographic Experts Group, is the name of the committee that created the standard. It is an ISO/IEC group of experts that was organized in 1986. In computing, JPEG is a commonly used method of compression for photographic images. The degree of compression can be adjusted, allowing a selectable trade off between storage size and image quality.

The JPEG standard specifies only how an image is transformed into a stream of bytes. JFIF (JPEG File Interchange Format), another standard developed by the JPEG Group, specifies how to create a file suitable for computer storage and transmission from a JPEG stream. This compression method is usually lossy, which means that some original image

information is lost and cannot be restored. However, this standard also allows lossless compression. The JPEG standard was developed to compress still and gray scale pictures, real world scenes and natural images. However, its performance in images with big discontinuities whether in color or in shades of gray is low.

JFIF encoding process usually starts with the conversion of the image color space from RGB to a color space consisting of one component that represents brightness, and two components representing chrominance, such as YCbCr or YUV. The image is then split into blocks of 8×8 pixels, and for each block, each of the Y, Cb, and Cr data undergoes a discrete cosine transform (DCT). The amplitudes of the frequency components are quantized. Regarding that the human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation, this allows a reduction in the amount of information in the high frequency components. If an excessively low quality setting is used, the high-frequency components are discarded altogether. This is done by simply dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer. This is the main lossy operation in the whole process. The resulting data of all 8×8 blocks is further compressed with a lossless algorithm. The image components are scanned in a "zigzag" order applying a run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length coding zeros, and then using Huffman coding on what is left to finish the compressing method.

JPEG standard possesses four different compression methods: sequential, progressive, hierarchic and lossless. In the sequential mode, each color component is encoded at once, whereas in both progressive and hierarchic mode, image components are encoded in multiple scans. These three mentioned methods are lossy and, therefore, it isn't necessary to profound them.

The JPEG lossless mode is based on a predictor that estimates the value of the actual pixel based on the values of the neighboring pixels. The value that is going to be encoded is the difference between the estimated value by the predictor and the value of the pixel that is being scanned. JPEG provides seven linear predictors, as shown in Table 2.2.

Generally, the performance of the several predictors may vary considerably from image to image. If encoding time is not a problem, then all of them can be tested and the one with the best compression rate chosen. This mode preserves the image, however it has lower compression rates.

	c	b	
	a	**X**	

Mode	Predictor
1	a
2	b
3	c
4	a + b - c
5	a + (b - c)/2
6	b + (a - c)/2
7	(a + b)/2

Table 2.2: On the left side, the actual pixel marked as "**X**" and its neighboring pixels: a, b and c. On the right side, the table with the seven Prediction Modes of JPEG standard.

2.6.2 The JPEG-LS standard

JPEG-LS [12, 15] is the state-of-the-art International Standard for lossless and near-lossless coding of continuous tone still images. It has been developed by the Joint Photographic Experts Group (JPEG) with the aim of providing a low complexity lossless image standard that could be able to offer better compression efficiency than lossless JPEG. The core of JPEG-LS is based on the LOw COmplexity LOssless COmpression for Images (LOCO-I) [14] algorithm, that relies on prediction, residual modeling and context-based coding of the residuals. Most of the low complexity of this technique comes from the assumption that prediction residuals follow a two-sided geometric probability distribution (TSGD) and from the use of Golomb codes, which are known to be optimal for this kind of distributions. Besides lossless compression, JPEG-LS also provides a lossy mode where the maximum absolute error can be controlled by the encoder. The basic block diagram of JPEG-LS is given in Fig. 2.9.

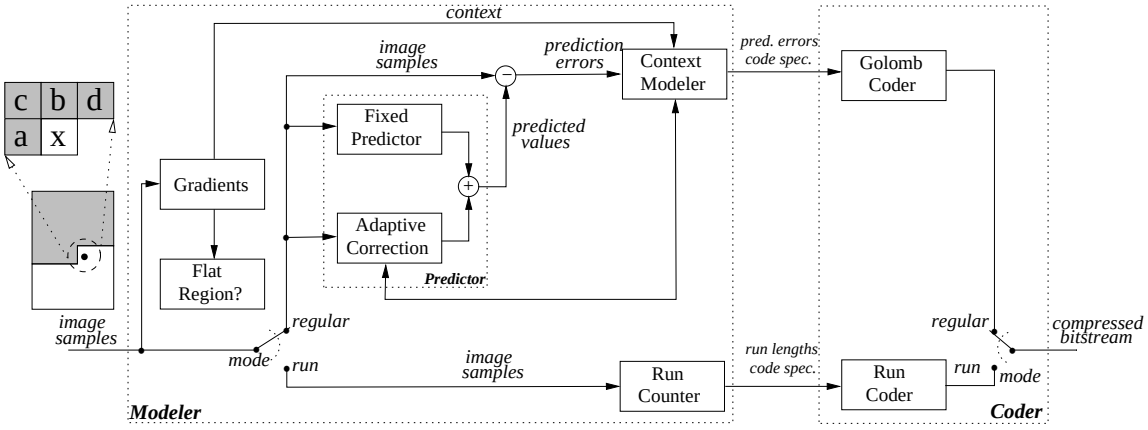

Figure 2.9: JPEG-LS block diagram [16].

The prediction and modeling units in JPEG-LS are based on the causal template depicted in Fig. 2.10, which is also known as MED predictor.

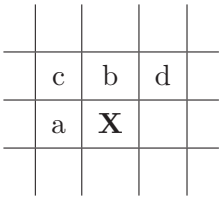

Figure 2.10: The pixel marked as "**X**" denotes the current sample, and a, b, c and d are the neighboring pixels.

The MED predictor uses, instead of a linear predictor like the JPEG standard, a nonlinear predictor given by

$$\widehat{\mathbf{X}} = \begin{cases} min(a,b) & \text{if } c \geq max(a,b) \\ max(a,b) & c \leq min(a,b) \\ a+b-c & \text{otherwise} \end{cases}$$

The predictor switches between three simple predictors: it tends to choose b in the case where a vertical edge exists on the left of the current location, a in cases of a horizontal edge above the current location, or $a+b-c$ if no edge is detected. The latter choice would be the value of $\mathbf{X}$ if the current pixel belongs to the plane defined by the three neighboring pixels with heights a, b and c. This expresses the expected smoothness of the image in the absence of edges.

LOCO-I algorithm shows relatively simplicity by assuming that the global statistics of residuals from a fixed predictor in continuous tone images are well modeled by a two-sided geometric distribution (TSGD) centered at zero. However, for context-conditioned predictors, TGSG is centered in an offset and hence, JPEG-LS will build those offsets using the following differences: $g_1 = d - b$, $g_2 = b - c$ and $g_3 = c - a$.

In order to use Golomb codes, the TSGD has to be first mapped into one-sided geometric distributions. In JPEG-LS, this mapping is done using

$$M(\epsilon) = 2|\epsilon| - \mu(\epsilon),$$

where function $\mu(\epsilon) = 1$ if $\epsilon < 0$ or 0 otherwise.

JPEG-LS has a run mode, when $a = b = c = d$, which indicates a flat zone.

2.6.3 The JPEG-2000 standard

JPEG2000 [12, 17] is the most recent international standard for still image compression (Part 1 was published as an International Standard in the year 2000). This standard is based on wavelet technology and embedded block coding (EBCOT) of the wavelet coefficients, providing very good compression performance for a wide range of bit rates, including lossless coding. Moreover, JPEG2000 allows the generation of embedded codestreams, meaning that from a higher bit rate stream it is possible to extract lower bit rate instances without the need for re-encoding.

This compression system allows great flexibility, not only for the compression of images but also for the access into the compressed data. The codestream provides a number of mechanisms for locating and extracting data for the purpose of retransmission, storage, display or editing. This access allows storage and retrieval of data appropriate for a given application, without decoding.

The block diagram of the JPEG2000 encoder is illustrated in Fig. 2.11. The discrete wavelet transform (DWT) is first applied to the source image data. The transform coefficients are then quantized and entropy encoded, before forming the output codestream (bitstream). The decoder is the reverse of the encoder: the codestream is first entropy decoded, dequantized and inverse discrete transformed, thus resulting in the reconstructed image data.

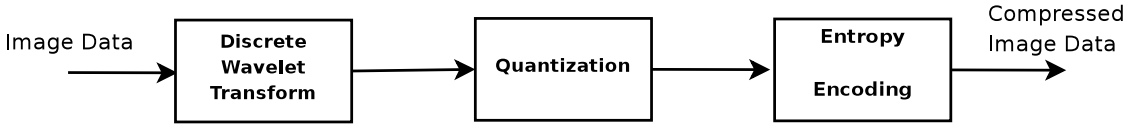

Figure 2.11: JPEG 2000: Block Diagram [12].

The source image data is first partitioned into rectangular nonoverlapping blocks, also known as tiles, which are compressed independently. Prior to computation of the discrete wavelet transform on each image tile, all samples of the image tile component are DC level shifted by subtracting 2^{b-1} to the pixels, where b is the component depth. DC level shifting is performed on samples of components that are unsigned only. Hereafter, an optional step can be also placed into the image: a component transformation. It is intended to undo the correlation between the colors and there are two types of transformation, depending on a requirement of a lossy or a lossless compression: Irreversible Component Transformation (ICT) and Reversible Component Transformation (RCT), respectively. The RCT performs an approximate transformation of the RGB space in YC_bC_r. These three pre-processing steps are depicted in Fig. 2.12.

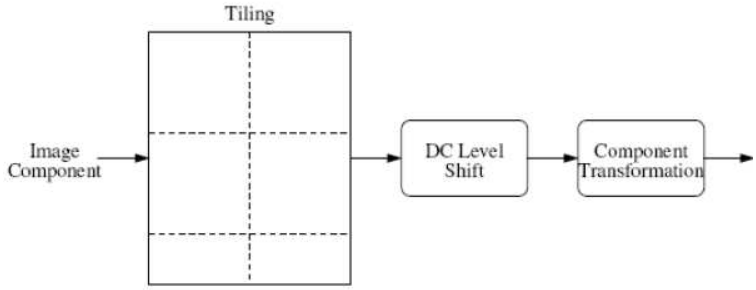

Figure 2.12: Pre-processing steps in the JPEG 2000 standard [12].

After these pre-processing steps, DWT is then applied. JPEG2000 uses two different types of wavelets: one for lossless (Le Gall 5/3 filter) and the other for lossy coding (Daubechies 9/7 filter). One of these filters is used in each tile, decomposing them into different levels. Those levels represent sub-bands, containing coefficients that describe the horizontal and vertical characteristics of the original tile. Since the DWT is unidimensional by nature, then, for bi-dimensional objects like images, DWT is applied in the horizontal and in the vertical directions, which result in four different blocks: one block with low resolution on both directions, one with high vertical resolution and low horizontal resolution, one with low vertical resolution and high horizontal resolution and one with high resolution on both directions. This process of applying DWT is then repeated a number of times on the low-resolution image block using the dyadic decomposition represented in Fig. 2.13.

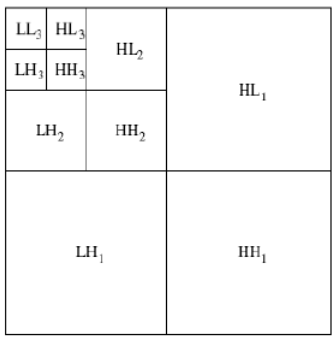

Figure 2.13: A representation of the dyadic decomposition, where L indicates low-pass filtering, whereas H means high-pass filtering [12].

After transformation, all coefficients are quantized. Quantization is the process by which the coefficients are reduced in precision. This operation is lossy, unless the quantization step is 1 and the coefficients are integers, as produced by the reversible integer 5/3 wavelet. After quantization, each sub-band is divided into rectangular blocks. Three spatially consistent

rectangles (one from each sub-band at each resolution level) comprise a packet partition. Each packet partition location is further divided into non-overlapping rectangles, called "code-blocks", which form the input to the entropy coder. The individual bitplanes of the coefficients in a code-block are coded within three coding passes. Each of these coding passes collects contextual information about the bitplane data. An arithmetic coder uses this contextual information and its internal state to decode a compressed bit-stream. These "code-blocks" are encoded independently from the others, which allows random access to the image content and efficient geometric manipulations. JPEG-2000 standard offers, comparing to JPEG or even JPEG-LS, a higher compression rate but, this better performance is due to a higher complexity of the algorithm.

2.6.4 The JBIG standard

The Joint Bi-level Image Experts Group (JBIG) [12] was issued in 1993 by the International Organization for Standardization / International Electrotechnical Commission (ISO/IEC) and Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) for the progressive lossless compression of binary images. The major advantages of JBIG over other existing standards, such as FAX Group 3/4, are its capability of progressive encoding and its superior compression efficiency. The term "progressive encoding" means that the image is saved in several "layers" in the compressed stream. When an image is decompressed and viewed, the viewer first sees an imprecise image (first layer) followed by improved versions (higher layers).

Even though JBIG was designed for bi-level images, it is possible to apply it to gray-scale images by separating the bitplanes and compressing each individually, as if it was a bi-level image. In this case, the use of Gray Code, instead of the standard binary code, may improve the compression efficiency.

The core of JBIG consists of an adaptive finite-context model followed by arithmetic coding. The context model used here relies on 1024 contexts when operating in sequential mode or on low resolution layers of the progressive mode, or 4096 contexts when encoding high resolution layers. For each pixel, JBIG examines a template made of the 10 neighboring pixels, marked as "X" and "A", and based on the value of these pixels choose the respective statistical model that will be used to encode the current pixel, marked as "?".

Figure 2.14 shows the two templates used for the sequential mode and for the low resolution mode. The encoder decides whether to use the three-line or the two-line template and indicates this choice in the bitstream (the two-line template results in a somewhat faster

	X	X	X	
X	X	X	X	A
X	X	?		

	X	X	X	X	X	A
X	X	X	X	?		

Figure 2.14: Templates for the lowest resolution layer. On the left, the three lines template. On the right, the two lines template [12].

execution and the three-line template produces slightly better compression). The template pixel labeled "A" is called *adaptive pixel* (AP). The encoder is allowed to use AP as a pixel outside the template and it uses two parameters in each layer to indicate the position of the AP in that layer.

More recently, a new version, named JBIG2 has been published [18], introducing additional functionalities to the standard, such as multipage document compression, two modes of progressive compression, lossy compression and differentiated compression methods for different regions of the image (e.g., text or halftones).

2.6.5 The PNG standard

Portable Network Graphics (PNG) [19] is an extensible file format for the lossless, portable, well-compressed storage of raster images. Color-indexed, gray-scale, and true color images are supported, with optional transparency (alpha channel). The images can have sample depths range from 1 to 16 bits.

PNG is designed to work well in online viewing applications, such as the World Wide Web and is robust, providing both full file integrity checking and simple detection of common transmission errors. Also, PNG can store gamma and chromaticity data for improved color matching on heterogeneous platforms.

Before compressing, a number of transformations are applied to the reference image to create the PNG image to be encoded. These transformations on the reference image will result in one of five types of a PNG image:

1. Truecolour with alpha: each pixel consists of four samples: red, green, blue, and alpha.
2. Grayscale with alpha: each pixel consists of two samples: gray and alpha.
3. Truecolour: each pixel consists of three samples: red, green, and blue. The alpha channel may be represented by a single pixel value. Matching pixels are fully transparent, and all others are fully opaque. If the alpha channel is not represented in this way, all pixels are fully opaque.

4. Grayscale: each pixel consists of a single sample: gray. The alpha channel may be represented by a single pixel value as in the previous case. If the alpha channel is not represented in this way, all pixels are fully opaque.

5. Indexed-color: each pixel consists of an index into a palette (and into an associated table of alpha values, if present).

A conceptual model of the process of encoding a PNG image can be described by the following steps:

- Pass extraction and scanline serialization;
- Filtering;
- Compression;
- Chunking;
- Datastream construction.

The palette and alpha table are not encoded in this way.

Pass extraction splits a PNG image into a sequence of reduced images where the first image defines a coarse view and subsequent images enhance this coarse view until the last image completes the PNG image. The set of reduced images is also called an interlaced PNG image. There are defined two interlace methods:

- Null method: pixels are stored sequentially from left to right and scanlines from top to bottom;
- Adam7: makes multiple scans over the image to produce a sequence of seven reduced images. In the first pass only 1 out of 64 pixels is transmitted, which results in a good approximation of the original image. It is this method that makes PNG work so well in web platforms.

After interlacing, each row of pixels, called a scanline, is represented as a sequence of bytes. Filtering is applied right after, which transforms the PNG image with the goal of improving compression. As a simple example, consider a sequence of bytes increasing uniformly from 1 to 255. Since there is no repetition in the sequence, it compresses either very poorly or not at all. But a trivial modification of the sequence, namely, leaving the first byte alone but replacing each subsequent byte by the difference between it and its predecessor, transforms

the sequence into an extremely compressible set of 255 identical bytes, each having the value 1.

Filters are applied to bytes, not to pixels, regardless of the bit depth or color type of the image. The filters operate on the byte sequence formed by the scanline that has been explained before. If the image includes an alpha channel, the alpha data is filtered in the same way as the image data.

PNG supports five types of filters, and an encoder may choose to use a different filter for each scanline in the image:

- None: each byte is unchanged;
- Sub: each byte is replaced with the difference between it and the "corresponding byte" to its left.
- Up: each byte is replaced with the difference between it and the byte above it (in the previous row, as it was before filtering).
- Average: each byte is replaced with the difference between it and the average of the corresponding bytes to its left and above it, truncating any fractional part.
- Paeth: each byte is replaced with the difference between it and the Paeth predictor of the corresponding bytes to its left, above it, and to its upper left.

The last method requires some explanation. Invented by Alan Paeth, the Paeth predictor is computed by first calculating a base value, equal to the sum of the corresponding bytes to the left and above, minus the byte to the upper left. Then, the difference between the base value and each of the three corresponding bytes is calculated, and the byte that gave the smallest absolute difference, that is, the one that was closest to the base value, is used as the predictor and subtracted from the target byte to give the filtered value. In case of ties, the corresponding byte to the left has precedence as the predicted value, followed by the one directly above. Note that all calculations to produce the Paeth predictor are done using exact integer arithmetic.

For all filters, the bytes "to the left of" the first pixel in a scanline shall be treated as being zero. For filters that refer to the prior scanline, the entire prior scanline and bytes "to the left of" the first pixel in the prior scanline shall be treated as being zeroes for the first scanline of a reduced image.

Unsigned arithmetic modulo 256 is also used in all filters, so that both the inputs and outputs fit into bytes.

The core of PNG's [12] compression scheme is a descendant of the LZ77 algorithm, known as the deflate algorithm. Deflate is comparable to LZW in both encoding and decoding speed and generally compresses better. In simplest terms, deflate uses a sliding window of up to 32 kilobytes, with a Huffman encoder on the back end. Encoding involves finding the longest matching string (or at least a long string) in the 32 KB window immediately prior to the current position, storing it as a pointer (distance backward) and a length, and advancing the current position and the window accordingly.

A marker bit in the final block identifies it as the last block, allowing the decoder to recognize the end of the compressed datastream.

To finish the encoding process, the compressed image is divided into conveniently sized chunks and inserted into the datastream.

2.7 Lossless Video Coding algorithms

There are a few methods proposed in the literature for lossless video coding, besides the H.264/MPEG4-AVC already described. The most important are the FFV1, HuffYUV and Lagarith coding methods.

2.7.1 FFV1

FFV1 [22], which stands for "FF video codec 1", is a lossless intra-frame video format. The encoder and decoder are part of the free, open-source library libavcodec in the project FFmpeg. FFV1 is included in ffdshow.

FFV1 is not strictly an intra-frame format. Despite not using inter-frame prediction, it allows the context model to adapt over multiple frames. This can be useful for compression due to the very large size of the context table, but can be disabled to force the encoder to generate a strictly intra-frame bitstream. During progressive scanning of a frame, the difference between a current pixel and its predicted value, judging by neighboring pixels, is sent to the entropy-coding process. The predicted value will be the mean value of a, b and $a+b-c$, according to the distribution shown in the Fig. 2.15. This predictor tries to estimate the local gradient in the image.

For improved performance and simplicity, the edges of the frame are assumed to be zero, to avoid special cases. The prediction in encoding and decoding is managed using a ring buffer.

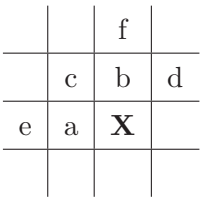

Figure 2.15: The causal template used in FFV1 for prediction.

The residuals are coded using either variable-length coding or arithmetic coding. Both options use a very large context model. There are two contexts models available. One of them uses a 3-order context model, based in the difference of a and c, c and b and b and d. The second is a 5-order context model and, besides the other three values, also uses the difference between f and b and between e and a.

2.7.2 HuffYUV

HuffYUV [23] is a lossless video compressor for AVI files written by Ben Rudiak-Gould. Source code is available and mostly distributed under the GNU General Public License. Its main goal is to temporarily store video data coming from a capture card for later editing. All the frames are encoded by intra-frame prediction so there are no inter-frame dependencies. HuffYUV's algorithm is roughly the same as lossless JPEG: it predicts each sample and the error is encoded using Huffman codes. This error is calculated by the following predictors, the same as shown in the Fig. 2.15:

- a;
- $a + b - c$, also known as the gradient;
- $Median(a, b, a + b - c)$.

The error signal in each channel is encoded with its own Huffman table. During compression, HuffYUV picks appropriate tables from its built-in collection. These tables are then indicated in the output file and used when decompressing. There are three different Huffman tables: YUV, RGB and RGB with decorrelation table. The channels of this last table are actually, R-G, G, and B-G. This yields much better compression than R, G, B.

The decoding process is the opposite of this process: the error value is extracted from the compressed Huffman stream. Then, the pixel value is reconstructed by computing the predictor value and adding it to the error value. The reconstruction of B and R values, when using the RGB with decorrelation, is made by just summing the value of G taken from the same pixel.

2.7.3 Lagarith

Lagarith [24] is a lossless video codec which offers better compression than HuffYUV, but worse than FFV1. However, Lagarith tends to be faster than these codecs. Lagarith is able to operate in several color spaces - RGB24, RGB32, RGBA, YUY2, and YV12. For DVD video, the compression is typically only 10-30% better than HuffYUV. However, for high static scenes or highly compressible scenes, Lagarith significantly outperforms HuffYUV. Lagarith is able to outperform HuffYUV due to the fact that it uses a much better compression method. Pixel values are first predicted using median prediction. Then, the bitstream may be subjected to a modified Run Length Encoding if it will result in better compression. The resulting bitstream from that is then compressed using Arithmetic compression, allowing the compressed size be very close to the entropy of the data. Additionally, Lagarith has support for null frames; if the previous frame is mathematically identical to the current, the current frame is discarded and the decoder will simply use the previous frame again.

Chapter 3

Color Quantization

In full color images, based on the RGB image color space, pixels are represented by 24 bits, 8 for each color component. Hence, a pixel can have one of about 16 million different colors available (2^{24}). However, sometimes this huge amount of colors can be too demanding. Moreover, in some cases, the image does not need to be so accurate, such as in web applications. Therefore, choosing an appropriate set of colors and to map the image into these colors might be a good option. This process of reducing the set of colors into a restrict number is called color quantization [26]. Two examples of color quantization can be seen in Fig. 3.1.

Figure 3.1: Two examples of color indexed images. Both images are the first frame from the video News, found at [3]. The picture on the left side is the full color image, which has been reduced to a palette of 8 colors (in the middle) and to 256, as it can be seen on the right side of the figure.

Usually, color quantization is made by taking the 256 most-common colors in the full color image. For each color in the input image, the algorithm searches for the closest color in the set of 256 colors, and displays that color instead. This algorithm transforms a full color image into a color-indexed one. Clearly, color quantization is a lossy process. It turns out that for most images, the details of the color quantization algorithm have more impact on

the final image quality than any errors introduced by compression algorithms, such as JPEG. Making a good color quantization method is a tough task and no single algorithm is best for all images.

There are several techniques available for reducing the number of colors in an image and, usually, these techniques take one of two possible approaches: *pre* or *post-clustering*.

3.1 *Pre-clustering* algorithms

Pre-clustering schemes are widely used and the first algorithms were based on this scheme. It simply divides the color space into a set of clusters and the centroids of these clusters define the resulting color map. Some of the most important algorithms which use this scheme are: popularity, median cut and octree.

3.1.1 Popularity algorithm

This algorithm [25] is implemented by first computing a color histogram of the image in RGB color space. Then, it simply chooses the K colors with the highest frequencies from the histogram, to yield a color palette. Its time and space complexity is high, and performs poorly on images with a wide range of colors, which are discarded due to low frequencies. Hence, the essential details cannot be preserved properly.

3.1.2 Median-cut algorithm

The Median-cut algorithm was originally described by P. Heckbert [27] and implementations are given by A. Kruger [28] and in the open source JFIF JPEG library. Its aim is to have each of the K output colors representing the same number of pixels in the input image. The starting point is the RGB cube that corresponds to the whole image, around which a tight-fitting cube is placed. The cube is then cut at the median of the longest axis and hence the name of the algorithm. This ensures that about the same number of colors is assigned to each of the new cubes. The procedure is recursively applied to the two new cubes until K cubes are generated and the centroids of the cubes become the K output colors. This algorithm performs better results than the popularity algorithm and, if properly implemented, it is also quite fast.

3.1.3 Octree algorithm

The algorithm proposed in [29] relies on a tree structured partitioning of the color space, where the root of the tree represents the entire space. As its name implies, an octree is an 8-way tree, that is, each node has up to eight leaf nodes. The main idea is that, after scanning the whole image, the color octree can have as many as the number of pixels in it. Each unique color would be represented by a leaf node in the tree. To get the number of colors down to K, the tree must be reduced by somehow merging colors. Colors that are very close together, that is, leaf nodes that share a parent, will be combined into a single average color. Those close colors will be deleted, and the new average color inserted into the tree. This will be repeated until the tree contains K colors. This implementation is improved by the following way: instead of building an entire octree containing all image colors, the tree can be reduced immediately whenever the number of leaves exceeds K. Thus, there are never more than K leaf nodes in the tree, saving considerable memory. This method has the advantage that only up to K colors need be stored at any time, which gives good efficiency both in time and space. However, the complicated merging operations may distort the essential details.

3.2 *Post-clustering* algorithms

The *Post-clustering* schemes consist on defining an initial selection of a palette followed by iterative refinement of it. These schemes may be computationally intensive due to the unlimited number of iterations. Some of the most important algorithms which use this scheme are: K-means, local K-means and NeuQuant.

3.2.1 K-means algorithm

This algorithm [30] starts by choosing K cluster centers randomly. During the first iteration, all the colors of the original image are assigned to the cluster whose center is the closest to the color and then, the cluster center will be recalculated. Following iterations will happen until it converges to a local minimum solution, that is, until the cluster centers of the K generated clusters do not change from one iteration to another.

The number of iterations required by the algorithm depends on the distribution of the color points, the number of requested clusters, the size of the color space and the choice of the initial clusters centers. Therefore, the computation of this algorithm can be very time consuming.

3.2.2 Local K-means algorithm

The local K-means algorithm [30] is a technique that approximates an optimal palette using multiple subsets of image points and it is based on a combination of the K-means algorithm and the Kohonen self-organizing map. The aim of this method is to simultaneously minimize both the TSE (Total Squared Error) and the standard deviation of squared error of pixels.

Let an image I be an array of M pixels (x, y). Then, $c_{(x,y)}$ is the color of each image pixel and $q(c_{(x,y)})$ is the quantized pixel. The total squared error is given by

$$\epsilon_{q(C,I)} = \frac{1}{M} \sum_{(x,y)\in I} \|c_{(x,y)} - q(c_{(x,y)})\|.$$

Small values of $\epsilon_{q(C,I)}$ guarantee that the quantization process accurately represents colors of the original image. However, it can be also misleading if an image has a non-uniformly distribution in the color space histogram. Thus, only with the total squared error function, less frequent colors in the image will be lost.

In order to avoid losing the information of less frequent colors, local K-means also tries to minimize the value of the standard deviation of squared error of pixels, which is given by

$$\sigma = \sqrt{\frac{\sum_{(x,y)\in I} \left(\|c_{(x,y)} - q(c_{(x,y)})\| - \epsilon_{q(C,I)}\right)^2}{M}}.$$

Minimizing both approximation measures simultaneously, the K-means algorithm will be able to represent the most frequent colors of the original image and, at the same time, to preserve variations of colors in the quantized image.

However, $\epsilon_{q(C,I)}$ and σ are image dependent measures. They treat each pixel independently and hence, spatial correlations among colors are not taken into account.

3.2.3 NeuQuant neural-net image quantization algorithm

This algorithm [30] has been developed to improve the median cut algorithm. It operates using a self-organizing Kohonen neural network, which self-organizes through learning to match the distribution of colors in an input image. NeuQuant algorithm offers a high-quality colormap in which adjacent colors are similar. By adjusting a sampling factor, the network can produce either extremely high-quality images slowly or good images within reasonable times.

3.3 Inverse color-mapping algorithms

Those techniques detailed above try to find the best color table for the quantized image. Once the color table is selected, the image must be scanned and all pixels assigned to an index in the color table. This requires an inverse color map. Transforming the 24 bits of a RGB pixel into an index table will require too many memory because the table will have approximately 16 million entries. Likewise, the trivial method performed by calculating the distance between each pixel of the image and the K colors of the palette to compute the smallest is impractical because it requires too much computation. In order to achieve an efficiently performance with the inverse color map, several methods have been designed to optimize the search of the closest representative. Examples of those methods are some improvements of the trivial method, the Locally sorted search and the three- and the two-dimensional Voronoi algorithm.

3.3.1 Improvements of the trivial inverse colormap method

Improvements of the trivial inverse colormap [30] algorithm are numerous. Poskanzer proposed improving the search by using a hash table, avoiding the search of the nearest representative for any color already found. However, this optimization is still inefficient, specially for images with a large number of colors, such as outdoor scenes. Another inconvenient is that this method can not be used after a *post-clustering* color quantization method, because the hash table must be recomputed after every iteration.

Another approach is to use a less expensive distance metric. Chaudhuri *et al.* proposed the L_α norm as an approximation of the Euclidean distance, with the L_α norm of a color $\mathbf{c}$ being defined by

$$\|\mathbf{c}\|_\alpha = (1-\alpha)\,\|\mathbf{c}\|_1 + \alpha\|\mathbf{c}\|_\infty$$

$$= (1-\alpha)\sum_{j=1}^{3}|\mathbf{c}^j| + \alpha \max_{j\in\{1,2,3\}}|\mathbf{c}^j|.$$

According to experiments performed by Verevka [31], the L_2 norm significantly speeds up the search and the introduced misclassification do not noticeably influence the quality of the input image.

The search cost can be further reduced using the following considerations:

- Calculation of the partial sum: the partial sum should be compared to the current

minimal distance before each new addition. The distance calculation finishes if the partial sum is greater than the current minimal distance.

- Sorting on one coordinate: the palette colors are sorted using one of the coordinates. The search starts with the palette entry whose first coordinate is the closest to the input color and continues in the increasing first coordinate distance order. The process finishes when the first coordinate distance between the next palette entry and the input color is greater than the current minimal distance.

- Nearest neighbor distance (NND): the search for the nearest color should finish when the current minimal distance is less than one half of the distance from the current palette color to its closest palette neighbor.

Using the above considerations, the algorithm's performance is improved, having a faster computation.

3.3.2 The locally sorted search algorithm

The locally sorted search algorithm [30] has been developed by Heckbert [27] and it is based on a uniform decomposition of the RGB cub into a lattice of N cubical cells, whose optimal number is still difficult to estimate. Each cell of this lattice contains the entries of the palette which could be the nearest representative of a color inside the cell. Each cell's list is defined by computing the distance r between the palette entry closest from the center of the cell and the farthest corner of the cell. Therefore, any palette entry whose distance to the cell greater than r is not included on the list. Given an input color $\mathbf{c}$, Heckbert's method determines which cell contains $\mathbf{c}$ and traverses its associated list to determine its closest palette entry.

This algorithm has a good performance when the palette entry is large and also when the input image has a wide range of colors. Therefore, the preprocessing time to build the cell's list is worth by the savings in search time.

3.3.3 Inverse colormap operation using a three-dimensional Voronoi diagram

The method described in [30, 32] computes the palette entries according to a three-dimensional discrete Voronoi diagram. This diagram is a special kind of decomposition of

a metric space determined by distances to a specified discrete set of objects in the space. Applying this diagram to an image defined by a RGB color space, a set of K colors belonging to the colormap are defined as Voronoi sites. Each site has a Voronoi cell, consisting of all RGB colors closer to a given palette entry than to any other. This will split an image into a set of K cells, all pixels of one cell being closer to one palette entry than the other.

The Voronoi diagram is encoded with the help of a three-dimensional array of integers, which represents a RGB color and contains the index of its closest representative. Hence, during a scan of an input image's pixel, its RGB value will be the index of the three-dimensional array and it will be assigned to that pixel the value contained in that array index. The main advantage of this method is that, once the three-dimensional array has been computed, the color index in the palette is retrieved without any computation. However, this algorithm involves the computation of all the displayable colors, which is useless if an image does not have represented all the colors of the RGB space. Besides that, it will be required to store 256^3 indexes.

3.3.4 Inverse colormap operation using a two-dimensional Voronoi diagram

This method has been proposed by Brun [33] and its main idea is to approximate the three-dimensional image color space into one of its two-dimensional projections so as to perform the inverse colormap operation with a two-dimensional Voronoi diagram instead of a three-dimensional. This method has been inspired by the fact that up to 99% of an image information is contained in the plane defined by the first two eigenvectors of the covariance matrix, as it was shown up by Ohta [34]. Moreover, the transformation matrix from the original color space to the eigenvector system is orthogonal.

Given an image color set, the first two eigenvectors of the covariance matrix are computed and then, an operator p is defined onto the plane P_{princ}, which is defined by these two eigenvectors. Given a three-dimensional Voronoi diagram associated with the colormap $\{c_1, \ldots, c_K\}$, it is then approximated by a two-dimensional diagram defined V_I by the sites $\{p(c_1), \ldots, p(c_K)\}$.

The computation V_I implies going through the input image twice: first, the three-dimensional distances are approximated by two-dimensional ones. In the second step, each projected color *p(c)* is rounded to the nearest pixel V_I.

These two steps often fail to map the input colors into their closest representatives. However, the errors caused are minor since the errors often affect the indexes of adjacent Voronoi cells. In order to correct those errors, it is evaluated, during the computation of the two-

dimensional Voronoi diagram V_I, the associated Delaunay graph D_I. These two data structures are then combined in the following manner: given an input color c the index V_I $[p(c)]$ is read in V_I and then, the set D_I $[V_I$ $[p(c)]]$ containing V_I $[p(c)]$ and the index of its adjacent cells it is read from D_I. The color c is then mapped to its closest representative among D_I $[V_I$ $[p(c)]]$,

$$Q(c) = argmin_{i \in D_I[V_I[p(c)]]} \|c_i - c\|,$$

where c_i denotes the representative color of the palette $\{p(c_1), \ldots, p(c_K)\}$ and $\|c_i - c\|$ denotes the Euclidean norm of the 3D vector c_i - c.

Note that $\mathbf{Q}(c)$ is computed from the 3D distances. The 2D Voronoi diagram is only used to restrict the number of distance computations.

3.4 Palette-reordering algorithms

After a limited number of K colors have been chosen to represent a true color image and all pixels have been assigned an index according to the entry of their representative in the color table, there is another technique which increases the compression of color indexed-images. Palette reordering has shown to be very effective at aiming that goal. An example of palette reordering can be seen in Fig. 3.2.

Figure 3.2: Two images of indexes of the same frame after color quantization. On the left, it is presented the image without palette reordering and on the right, the same image after the reordering process.

Since the mapping between index values and available colors is not unique, a proper indexing of the color mapping can be achieved by permuting the palette color entries, under the condition that the corresponding index image is changed accordingly. This new assignment is based on the global characteristics or statistics of the image. There are various palette

reordering algorithms that have been developed. Those algorithms can be divided in two groups: color-based and index-based methods.

3.4.1 Color-based Methods

Color-based methods rely only on the information provided by the color palette. Examples of this type of palette reordering method are the Intensity-based method and Po's Method.

Intensity-based method

This method [35] is the simplest among the color-based methods and one of the simplest among all the reordering methods. Like its name denounces, this method reorders by the pixel's intensity, that is, its luminance. Being proposed by Zaccarin *et al.*, it relies on the assumption that, if a given pixel has neighbors of similar luminance, then colors with similar luminance should have similar indexes.

For instance, if the image is based on a RGB color space, for each color, it will be calculated its luminance, according to

$$Y = 0.299R + 0.587G + 0.114B,$$

where R, G and B denote the intensities of the red, green and blue components, respectively. Then, all the colors in the palette will be sorted according with their luminance values. The main advantage of this method is its fast computation, being the fastest one amongst all the palette-reordering methods.

Po's method

Also referred to as the "closest pairs ordering", Po *et al.* [36] proposed this reordering technique, with the aim of assigning close indexes to colors that are close in three-dimensional (3-D) color space. This algorithm starts by assigning index zero to the closest color to the origin of the color space and then proceeds by finding the color that is the closest to the color corresponding to the previous index and assigns the current index to it. This algorithm is a simple technique. However, it generally generates poor solutions [35].

3.4.2 Index-based Methods

Another group of palette reordering methods are based on the indexes information. Their main idea is that colors that occur frequently close to each other should have close indexes. Therefore, based on this principle, the assignment of the indexes is usually guided by measuring the number of occurrences of a pixel that is spatially adjacent to another pixel, according to some predefined neighborhood. Examples of this type of methods are Memon's Method, Zeng's Method and Modified Zeng's Method.

Memon's Method

Memon *et al.* formulated the problem of palette reordering within the framework of linear predictive coding [37]. In that context, the objective is to minimize the zero-order entropy of the prediction residuals. They noticed that, for image data, the prediction residuals are often well modeled by a Laplacian distribution and hence, minimizing the absolute sum of the prediction residuals leads to the minimization of the zero-order entropy of those residuals. For the case of a first-order prediction scheme, the absolute sum of the prediction residuals reduces to

$$E = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} N(i,j)|i-j|,$$

where N(i,j) denotes the number of times index i is used as the predicted value for a pixel whose color is indexed by j, and M denotes the number of colors of the image. The problem of finding the bijection that minimizes E can be formulated as the optimization version of the optimal linear ordering problem, which is known to be NP-complete [37].

The so-called pairwise merge heuristic is one of the Heuristics proposed by Memon *et al.* for finding a good solution to the above stated problem. The main idea is to repeatedly merge ordered sets of colors until obtaining a single reordered set. Initially, each color is assigned to a different set. Then, the two sets, S_a and S_b, maximizing

$$\sum_{i \in S_a} \sum_{j \in S_b} \big(N(i,j) + N(j,i)\big)|i-j|$$

are merged together. This procedure should be repeated until having a single set. To alleviate the computational burden involved in selecting the best way of merging the two sets, only a limited number of possibilities are generally tested [37]. Amongst all the methods described

in this section, Memon's time computation is the slowest one but, on the other hand, it has the highest average compression improvement.

Zeng's Method

Zeng [38] proposed a one-step look-ahead greedy approach, which begins by finding the index that is most frequently located adjacent to the other different indexes and the index most frequently found adjacent to it. This pair of indexes is the starting base for an index list P_n that will be constructed, one index at a time, during the operation of the reindexing algorithm. This process will be iterative and new indexes will only be attached to the left or to the right extremity of the list. Before the first iteration, P_0=(l_0, l_1), where l_0 and l_1 are the starting base mentioned above. During the following iterations, the new unassigned index u_i to be attached will be the one which satisfies:

$$u_L = argmax_{u_i} \cdot D_L(u_i, N), \tag{3.1}$$

where

$$D_L(u_i, N) = \sum_{j=1}^{N} w_j \cdot C(u_i, l_j),$$

and

$$u_R = argmax_{u_i} \cdot D_R(u_i, N), \tag{3.2}$$

where

$$D_R(u_i, N) = \sum_{j=1}^{N} w_{N-j} \cdot C(u_i, l_j)[35].$$

The function $C(i, j) = C(j, i)$ denotes the number of occurrences, measured in the initial index image, corresponding to pixels with index i that are spatially adjacent to pixels with index j. The w_j are weights controlling the impact of the $C(i, j)$ on $D_L(u_i, N)$ and $D_R(u_i, N)$, and the sums are performed over all the N indexes already located in the next list $P_N = (l_1, l_2, \ldots, l_N)$. The new index list will be given by $(u_L, l_1, l_2, \ldots, l_N)$ if $D_L(u_i, N) > D_R(u_i, N)$ or by $(l_1, l_2, \ldots, l_N, u_R)$ otherwise. Finally, the indexes in the list are relabeled, creating $P_{N+1} = (l_1, l_2, \ldots, l_{N+1})$ and a new iteration starts, until all indexes are attached to P_N.

Then, the reindexed image is constructed by applying the mapping $l_j \mapsto j-1$ to all image pixels, and changing the color-map accordingly.

Finally, Zeng *et al.* [38] suggested that a reasonable choice for the weight w_j is given by

$$w_j = log_2\left(1 + \frac{1}{j}\right),$$

where j corresponds to the distance between the current left end position of the index list and the position of index l_j in the index list.

Modified Zeng's Method

This modified version has been developed by Pinho and Neves [39], where their theoretical analysis of Zeng's method for the case of Laplacian distributed differences of neighboring pixels lead to a set of parameters that differs from the one originally suggested by [38] and explained previously. It is suggested that, under that Laplacian model, the process of building P_{N+1} from P_N should be conducted in two steps. First, the index u satisfying 3.1 (or 3.2) is determined using the weights w_j (or w_{N-j}) all having the same value. Then, the correct side in P_N where the new index u should be attached is determined based on the sign of

$$\Delta = \sum_{j=1}^{N} (N + 1 - 2j)\, C(u, l_j).$$

If $\Delta > 0$, the left-side should be chosen, otherwise choose the right-hand side of P_N. Experiments made by Pinho and Neves shown that, usually, the Modified Zeng's Method has a better performance than Zeng's Method.

3.5 Dithering

After choosing the best set of K colors and after assigning to all pixels of the image the proper indexes of the colormap, there are still many images that look bad. They have visible contouring in regions where the color changes slowly. However, the capability of the human visual system to distinguish different colors declines rapidly for high spatial frequencies and, therefore, the human visual system may produce additional perceived color by averaging colors in the neighborhood of each pixel. Thus, this eye redundancy can be exploited to enhance *a posteriori* quantization processes. It has been shown that excellent image quality with color palettes of very small size can be achieved with this basic idea. A variety of techniques have been proposed such as noise and ordered dithering or error diffusion techniques. However, those techniques have a disadvantage: since they consider each color component an individual gray scale image, color shifts and false textural contours appear on the resulting image, due to the correlation between the color components.

3.5.1 Noise and Ordered Dithering

The main idea of noise dithering is very simple: it consists on the addition of a random number to each pixel of the original image. However, this process introduces some "noisy". In the ordered dithering technique, it is added a pseudo-random two-dimensional periodic dither signal to each color component of the pixel prior to finding its best match in the colormap [41]. The main disadvantage of these algorithms is that the processed images lose their natural aspect, because certain textures generated in the dithering process can be easily observed.

3.5.2 Error Diffusion Technique

Error diffusion algorithms build dithering patterns dynamically, based on error dispersion. One example of this technique is the Floyd-Steinberg Dithering algorithm, which is depicted in Fig. 3.3.

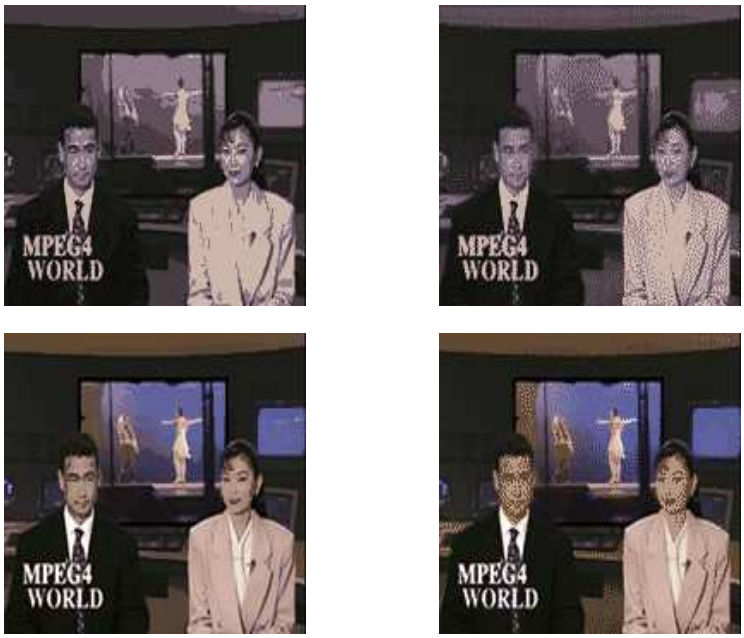

Figure 3.3: Floyd-Steinberg Dithering algorithm applied to the frames with a set of 8 and 256 colors.

This algorithm is one of the algorithms which generates the best results despite being computationally too slow. In fact, there are many variants of this technique as well, and the better they get, the slower they are. This algorithm starts by calculating the error between the original pixel and its representative in the palette, during the mapping process. This error

is distributed to neighboring pixels which have not been mapped yet. The Floyd-Steinberg method gives 7/16 to the next pixel to the right, 3/16 to the previous pixel on the next scanline, 5/16 to the adjacent pixel on the next scanline and 1/16 to the next pixel on the next scanline. An advantage of error-diffusion techniques over ordered dithering is that it can use an arbitrary colormap, allowing to be processed at the same time with inverse colormap algorithms.

3.6 Color quantization in video

All of those techniques explained and detailed in the sections above are used in images. However, those techniques can also be applied to video, although it has not been adequately exploited yet. One part of this thesis focuses on reducing the set of colors of each video frame, mapping them to each pixel and later reordering its palette. The way we implemented these three steps will be better explained in the next chapter of this thesis. One of the main problems that stood out during this process was that, between two consecutive frames, some flickering was notorious. This is due to a different assign of colors between two pixels that had the same original value, as shown in Fig. 3.4. Once they belong to different frames and each frame has different color statistics, their processes of quantization and inverse colormapping produce different results. One possible way to prevent this situation can be using an adaptive palette throughout the frames of the video.

Figure 3.4: An example of flickering, from the frames number 90 and 91 of the video News with a set of 256 colors. Here, it is notorious a difference of colors applied to the hosts faces from one frame to another.

Chapter 4

Proposed video coding algorithm

In this thesis, we developed two video coding algorithms: one using Golomb coding and the other based on Arithmetic coding. The bitstream generated for both encoders is basically the same and both of them will be described below.

4.1 The single plane video bitstream

The bitstream proposed in this thesis allows the storage of two different types of video: gray-scale and color-indexed. As depicted in Fig. 4.1, it is formed by a header, which contains some information about the video, followed by the data of the frame and, if the video is color-indexed, by the palette of colors.

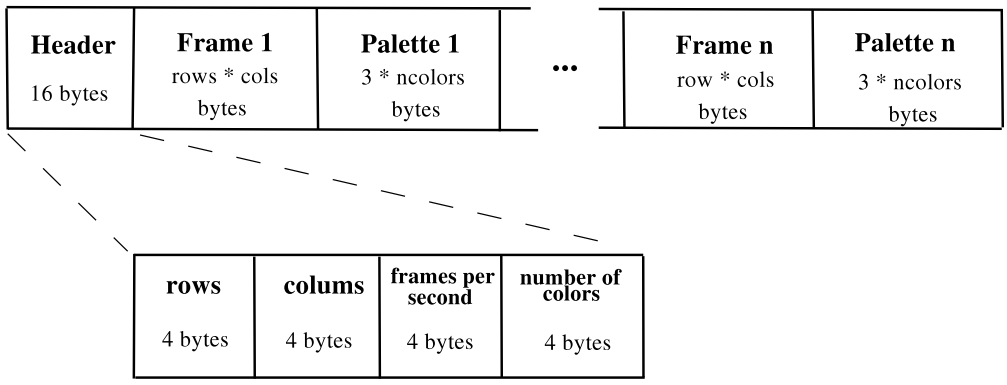

Figure 4.1: Overview of the bitstream before being sent to the encoder.

This bitstream is sent to the encoder and then a new bitstream is obtained. This new bitstream has also the header followed by the encoded frames. In color indexed videos, the palette is stored after its respective frame. However, it can be encoded before being stored

or being sent directly to the bitstream. This is due to the fact that, after some tests that we have made, we found out that the compression rate did not improve when encoding the palette with the Golomb codes. However, when encoding the palette with Arithmetic coding the compression rates have been slightly better than sending the palette directly to the bitstream. Due to these facts, it was decided that, for the Golomb codes, the palette is written directly to the bitstream, whereas when encoding with Arithmetic coding, it will be encoded as well as the rest of the data, as shown in Fig. 4.2.

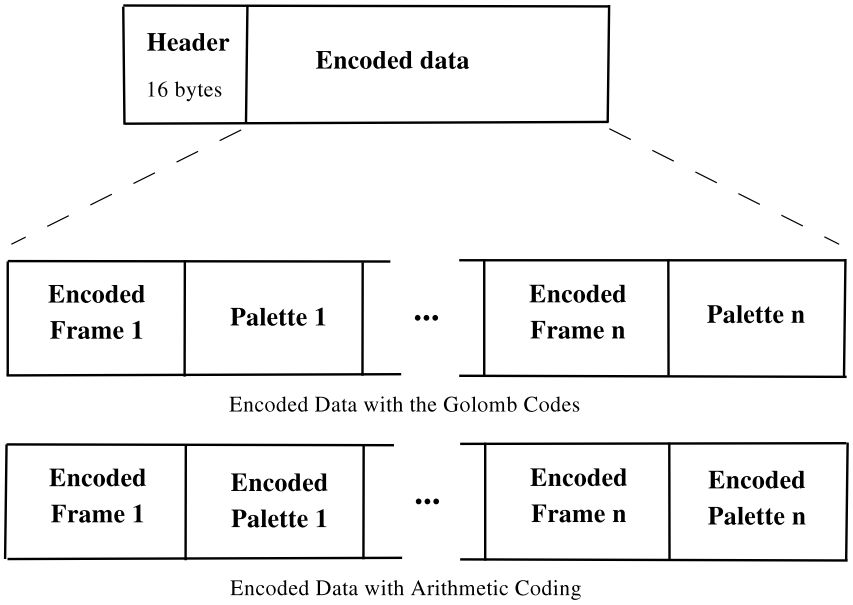

Figure 4.2: The bitstream after being encoded, using the Golomb codes or Arithmetic coding.

4.1.1 Header information

The header of the bitstream is composed by sixteen bytes, as depicted in the lower part of Fig. 4.1, each one containing a value represented in the decimal format. The first eight bytes contain the video dimensions, where the first four bytes represent the number of columns and the second the number of rows. The next value in the header is the number of frames that have been captured during the recording, divided by the elapsed time of the video, that is, the number of frames per second. This value is important when displaying the video, because without it there will not be any reference to display the image in the correct motion. Moreover, multiplying or dividing it will produce the effect of fast or slow motion, respectively. The last four bytes of the header indicate whether the video is color-indexed or gray scale. A non-zero value of n indicates that the stored data belongs to a video with a reduced set of n colors, otherwise it belongs to a video only characterized by luminance values (gray scale).

4.1.2 Frame structure

As the video format in this thesis is represented only by a single information plane, a single frame structure will occupy a total of $rows \times columns$ bytes, as indicated in the header. The image data is stored in the bitstream from left to right, top to bottom, and each byte represents one pixel of the frame. The value of each pixel can be its luminance or an index to its representative color in the palette, if the data represents a gray scale or a color indexed video, respectively. However, the main goal is to reduce the size using lossless coding and, therefore, it is expected that each pixel can be represented by less than one byte, otherwise the compression method is not efficient.

4.1.3 Color palette

When a video is color-indexed, the palette of colors is attached to the bitstream after its respective frame structure. This palette of n colors will have a size of $3n$ bytes. Each one of these n colors are represented in a RGB format and, therefore, three bytes are needed, one for each component (Red, Green and Blue). The palette is stored by writing the color components together, from the first to the last color.

Not all the palettes in the video sequence contain n colors. The program used to quantize the input image, based on the Median Cut algorithm, allows a previous definition of how many colors the output image will have. Nevertheless, this program might not choose a n colors palette for a given frame. That situation happens mostly in palettes with a set of 256 or more colors. In the other cases, all the palettes had the same number of colors. Using a sub-header in all the frames, indicating how many colors the palette contains, could be an idea. However, in videos with hundreds of frames, this approach might increase the size of the video in hundreds of bytes, when the palette has a reduced set of colors. Furthermore, in color-indexed videos resulting from a video with a wide set of colors, such as natural videos, certainly the quantization process will choose the n colors defined in the initial parameters. Hence, it was decided that all the palettes of the video should occupy the same size in the bitstream, and the indexes which store empty colors are assigned with value 0.

It is necessary to refer that the choice of a program that reduces the set of colors based on the Median Cut algorithm was easy, because it is one of the most simple algorithms available and the study of the performance of Color Quantization Methods was not a goal in this thesis.

4.2 Overview of the proposed method

The contents of the bitstream detailed above are preceded by several processes in order to transform the original video information into data that fits the single information plane proposed here (see Fig. 4.3). When dealing with medical videos, the process is simple, because the captured components contain only luminance information that is, the video is gray scale.

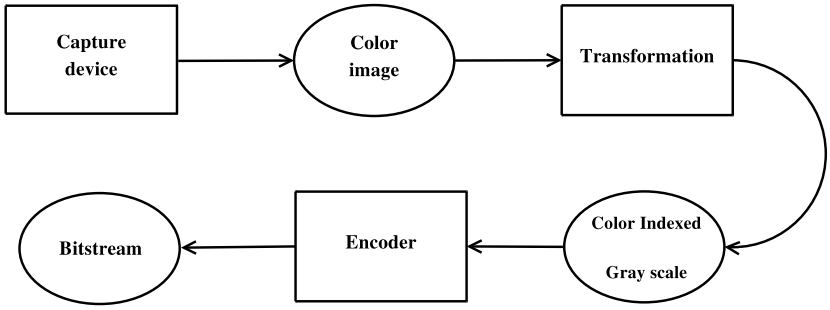

Figure 4.3: The procedure taken from the capture of a color image to the final step of encoding into the proposed bitstream.

However, if the video was captured by a digital camera, the transforming process will require more steps. A digital camera is a device that records the frames captured by a light sensitive sensor. It is formed basically by a lens, a color sensor, software for image treatment and hardware capable of processing and transmitting the captured images. The component in the digital camera that is responsible for capturing colors is the color sensor (CCD). Almost all the CCD sensors exploit the particularity of most of the colors of the visible spectrum could be reproduced by adding distinct parts of red, green and blue light. It consists of a square grid of capacitors, which are charged by a photosensitive element covered by a filter that only allows one component of light (red, green and blue) to reach the sensor. The image is formed measuring the charge of each capacitor by a control circuit.

For each pixel, the CCD does not acquire the information of the three components. Instead, a Bayer pattern is used, as depicted in Fig. 4.4. It is a repeating 2×2 mosaic pattern of light filters, with green ones at opposite corners and red and blue in the other two positions. The predominance of green takes advantage of properties of the human visual system, which determines brightness mostly from green information and is far more sensitive to brightness than to hue or saturation [40].

After capturing the video with a digital camera or a scanner, the video frames are represented by a RGB colorspace. The next step is to transform this RGB video into one according with the proposed model. The remaining process is different for color-indexed and for gray

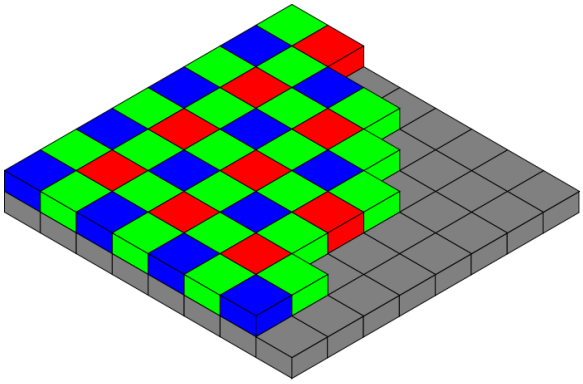

Figure 4.4: The Bayer pattern [42].

scale videos.

4.2.1 Adapting the data into a gray scale video

Gray scale videos do not have any color information, only luminance. It is the luminance that generates various shades of gray. Full luminance generates white and decreasing its value darkens the color until it reaches black. RGB is not the proper color space to handle this case because its three components are correlated. One possible color space which separates luminance from chrominance is YUV. Transforming a YUV video into a gray scale one is a simple process: it is only needed to maintain the luma component, Y, and discard U and V information. Thus, the following step is to transform the RGB video into a YUV format and then discard the U and V components. The relation between those color spaces is given by:

$$Y = 0.299R + 0.587G + 0.114B \tag{4.1}$$

$$U = -0.147R - 0.289G + 0.436B$$

$$V = 0.615R - 0.515G - 0.100B$$

In order to perform this transformation, it has been developed an algorithm in C code (see Annex B.4). The video file is loaded and all the video frames are scanned, from top to bottom, left to right. During the scan, the value of Y as a function of R, G and B components is computed for each pixel of the frame. After writing the header with the values of the rows, columns, frames per second and the zero value, indicating that the video is gray scale, the luminance values are dumped to the bitstream in the same order as the pixels were scanned.

4.2.2 Adapting the data into a color-indexed video

This process of transformation is more complex than to produce a gray scale video. Although the original video data is in the correct color space to perform the proper adaptation, it will require a few more steps. Since the color quantization algorithms are optimized to be applied to images, a process of splitting the video frame by frame is required. An algorithm has been developed to split all the video frames (Annex B.5). The program used to perform the image quantization only supports image files in a PPM format (Portable Pixmap Format) and it is included in the Netpbm library.

Hence, the developed algorithm performs a transformation of all the video frames into PPM image files. This is achieved by writing in all images a header containing a two-byte file descriptor, in ASCII, that explains the type of file. The descriptor is a capital P followed by a single digit number, which in this case, will be the number 6, indicating that it will be used 24 bits per pixel: 8 for red, 8 for green and 8 for blue. Then, in the second line, it is written the number of columns, rows and finally, in the third row, the highest value possible.

A cycle scanning all the video frames is performed and, during one cycle, it is open a new file where it is written the header explained above. The name of this new file is a concatenation between an initial parameter passed in the command line with the number of the frame that is being processed in this cycle. The pixels of the frame are scanned, from top to bottom, left to right, and their values of R, G and B are dumped in the PPM file in that order.

Now that all the frames are split, the program based on the Heckbert's "median cut" algorithm, *ppmquant*, is used to perform color quantization. However, performing this process to all the frames will take too long if it is done manually, calling the command to every single frame. In order to solve this problem, it has been developed a shell script in which *ppmquant* is called as many times as the number of frames to be processed (Annex B.16.1). This shell script performs as follow:

```
1. It asks how many colors are desired to the new color set;
2. It is read that value;
3. It asks what is the name of the new quantized images;
4. It is read that name;
5. It asks what is the name of the folder to save all the new quantized images;
6. It is read the name of the folder;
7. It is created the folder with the indicated name;
8. It is made a scan of all the images to be quantized;
```

```
9. A cycle from the first PPM file to the last one begins, where:
   9.1 For each PPM file, the quantization process is performed;
   9.2 Each PPM file will be stored in the given folder and its name will be
    a concatenation of the given name, the original image name and its
    respective frame number in the original video.
```

Note that this shell script is called inside the folder where all the original frames are placed but the shell script itself is not placed inside that folder. Otherwise, when scanning all the images to be quantized in the folder, the file containing the shell script would be considered another file to be quantized and it would report an error. The same happens with the folder to save the quantized images, being created outside the folder.

After having quantized all the frames, the next step is to assemble them and re-build the video again. Another algorithm has been developed and besides assembling all the frames together, this algorithm also builds the color table of each frame (Annex B.6). This program starts by opening a new file, where the bitstream will be dumped. The header of the bitstream is written in the file with the number of rows, columns and frames per second that were known previously, followed by the number of colors in the palette, which is given to the algorithm by the command line as a initial parameter. A cycle begins by scanning all the images to assemble. During a cycle, all the pixels of the frame are scanned and their three color components are assigned to the palette if they haven't been found on previous pixels, occupying the first available entrance of the palette. With the palette built, another scan pixel by pixel begins again in the PPM file. According to the value of the components of the pixel, it is made a search in the palette to find that values. When those values are found, the search stops and their index is assigned to the corresponding pixel in the frame structure. The files are closed and the algorithm ends.

After all these transformations, a color-indexed video represented by the proposed bitstream is accomplished. However, to achieve higher compression rates, a palette reordering can be made. We implemented a program (Annex B.7) in which the reordering algorithm is the fastest one, even though it is not the most efficient: reordering according to luminance, which was explained in 3.4.1. After sorting all the colors from the lowest to the highest value of luminance, an adjustment of the indexes stored in the frame structure must be made by scanning all the pixels of the frame to replace the old index value with the new one.

4.3 Encoding the bitstream

After having the video information in accordance with the proposed model, the next step is to encode all the data in a lossless mode. For this purpose, we developed two algorithms, using two diferent encoding methods: Golomb codes and Arithmetic coding, already detailed in Chapter 2. Both of the algorithms have basically the same encoding structure when dealing with the video frames. They encode the first frame in intra mode and the remaining frames in inter mode. The block diagrams of the intra and inter modes are depicted in Figs. 4.5 and 4.6, respectively.

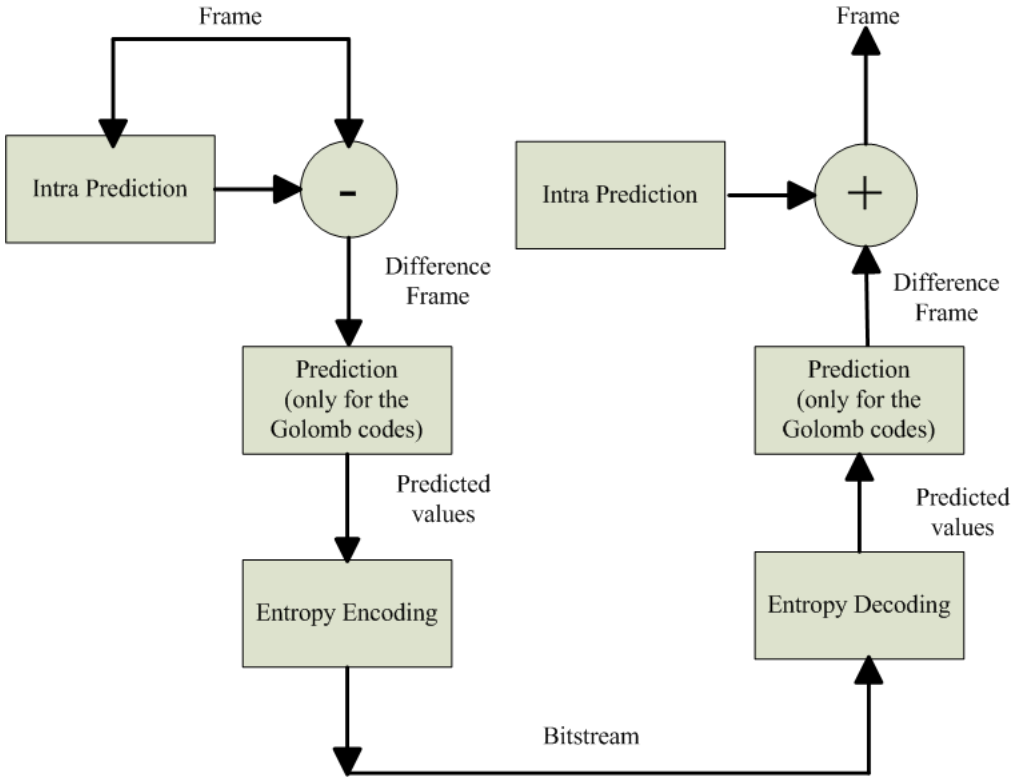

Figure 4.5: Block diagram of the Intra mode.

Intra mode prediction implies that the frames are encoded independently from the others. According to Fig. 4.5, the intra prediction is first applied, where the frame is scanned from top to bottom, left to right and the value of the pixels are subtracted by the value of the previous one and then sent to the next step of the encoding process. The only exception is the first pixel, which is sent with its original value to the encoder. If we are using the algorithm which encodes with the Golomb method, the residuals can have a middle step before being encoded: they can pass into another predictor, the MED predictor, which was previously explained in Section 2.6.2. Encoding is the last step before sending the data to the bitstream.

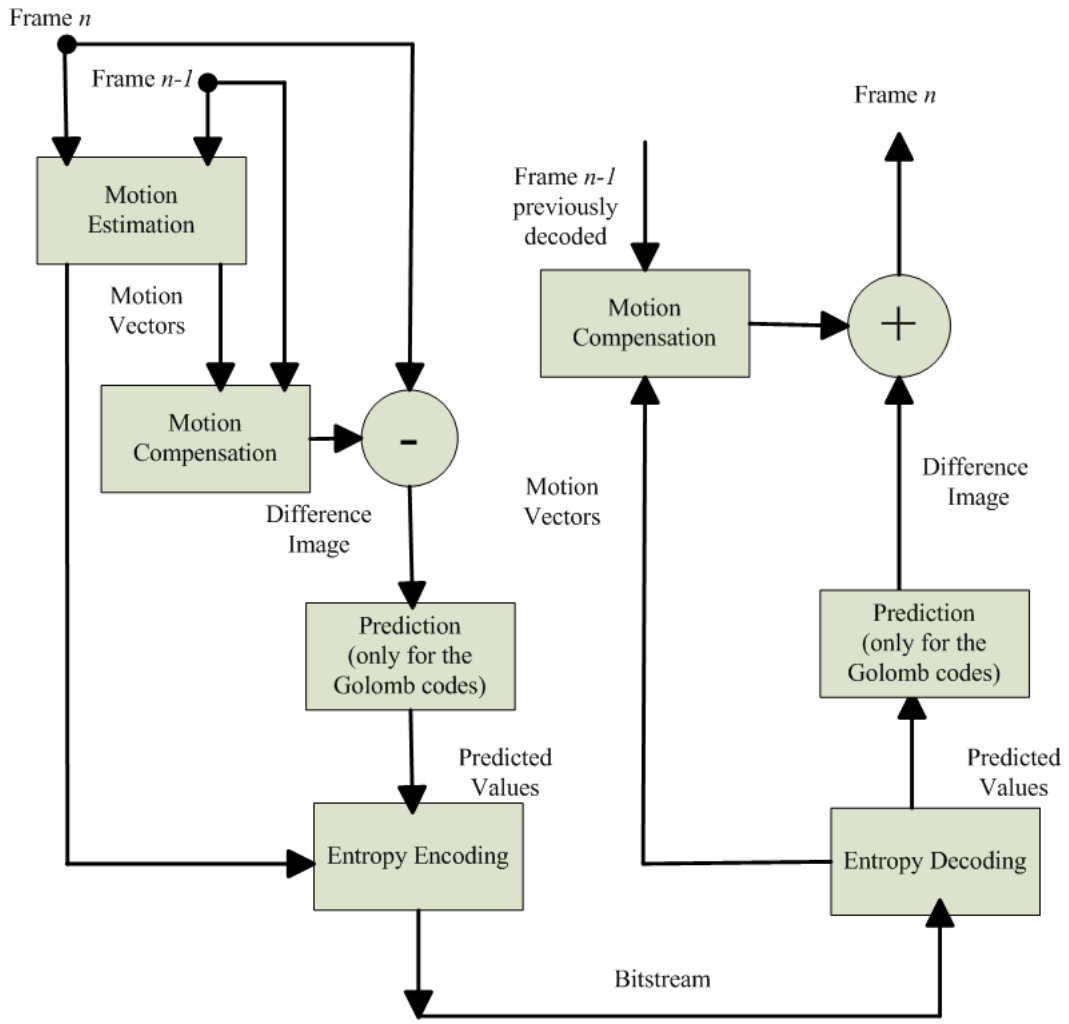

Figure 4.6: Block diagram of the Inter mode.

The decoding process of the intra mode is made by applying the encoding process backwards, as depicted on the left side of the figure.

In inter mode, it is necessary to have a reference frame. In this case, we decided that the reference frame should always be the previous one. Besides this option, another one valid would be the first frame of the video and the following ones in a range of K frames encoded in intra mode. In this case, the reference would always be the closest frame to the one that is being encoded in that moment.

The inter mode prediction starts by performing motion prediction, already explained in Section 2.3. The motion vectors are used to calculate the residuals of the frame and then sent to the encoder. As in the intra mode, these residuals can pass in the MED predictor before being sent to the encoder, when the algorithm uses the Golomb method. The decoding

process uses the encoding process backwards to restore the original data.

Basically, the process of encoding the video frames follows the mechanism depicted in Fig. 4.7.

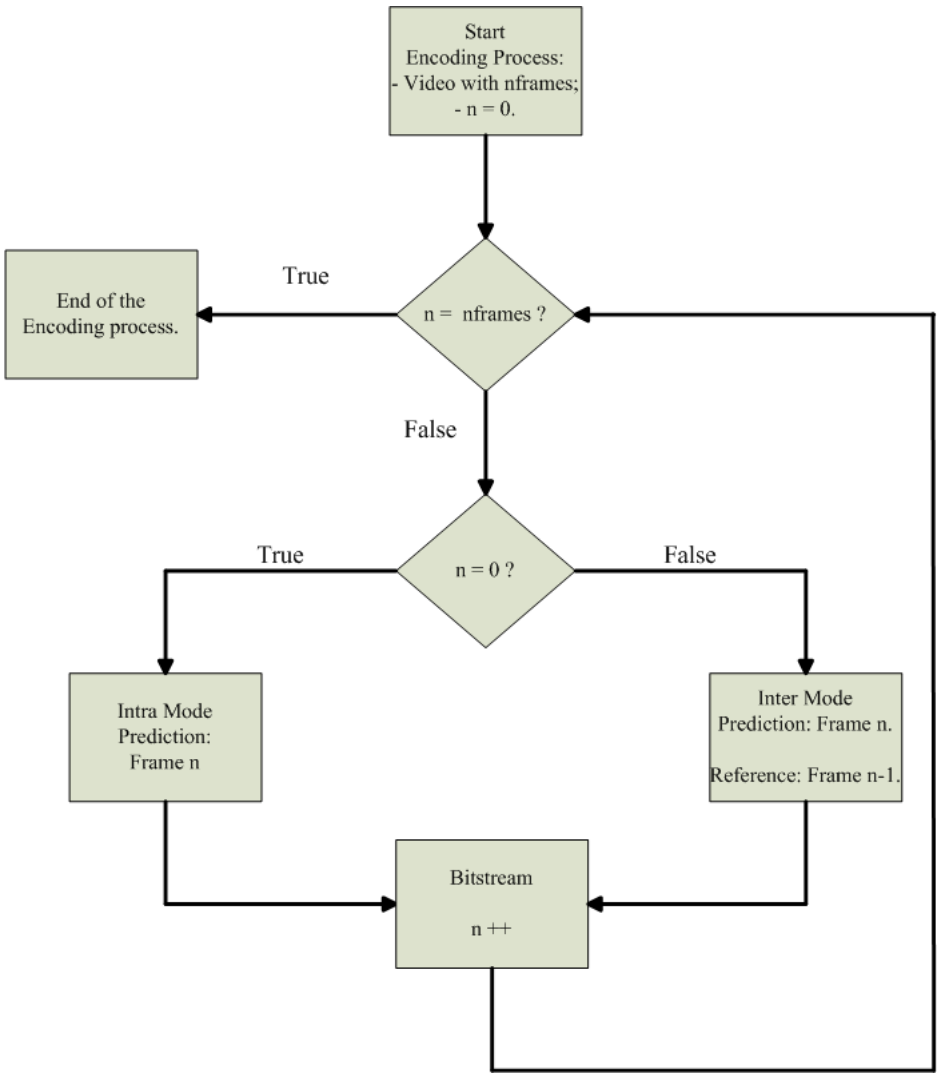

Figure 4.7: Diagram which reflects the mechanism used by both encoding methods follow.

Despite all these similarities in the encoding process, those methods have some differences between them. One of those differences is how they deal with the colormap when the video is color-indexed. This and other specific particularities of each method will be detailed next.

4.3.1 Encoding the bitstream with the Golomb codes

Besides encoding the data with a high compression rate, it is also necessary to assure that the decoding process will occur successfully. Therefore, it is necessary to write in the bitstream some information about the encoding process, which will be placed after the header.

This sub-header, which is depicted in Fig. 4.8, contains the following data: the number of bits to encode the residuals and the motion vectors, which is given by $\log_2(m)$ (Section 2.1), the size of the block, the interval to seek in the neighborhood around the block, and a flag to indicate if the MED predictor was used.

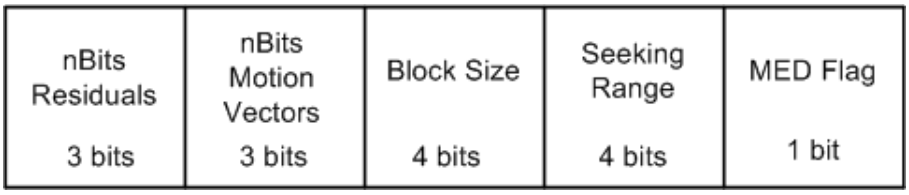

Figure 4.8: The sub-header stored in the bitstream containing the relevant encoding configuration values.

Moreover, before storing data from a determined frame and its respective colormap, a 16-bit marker flag is also written with the value 0xFFFF.

Once Golomb codes only process absolute values, it is necessary to distinguish whether a value is positive or negative. Hence, before dumping the result of encoding the absolute value, an one bit flag is sent to the bitstream: 1 if the value is negative and 0 if it is positive.

Finally, if the video is color-indexed, the palette is not encoded, being sent to the compressed file without any transformation, as it was already explained in Section 4.1.

4.3.2 Encoding the bitstream with the Arithmetic coding

This algorithm can use three different finite-context model orders: 0, 1 or 2. As well as in the algorithm with Golomb codes, it is necessary to write the size of the block and the seeking interval, along with the order of the finite-context model, in order to guarantee that the decoding process will occur successfully (Fig. 4.9).

Every time a new frame is sent to the encoder, all the values of the table of probabilities is set to 1. There will be three different table of probabilities: one for the residuals of the frame, one for the motion vectors and one for the palette.

For the residual values, this table has 511^{order} entries if it is a gray scale video or $2n-1^{order}$

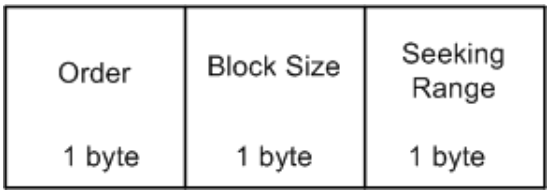

Figure 4.9: The sub-header stored in the bitstream containing the relevant encoding configuration values.

if it is a color-indexed video. These values require some explanation: as the values are predicted, they can be either negative or positive. In a gray scale video, which has values from 0 to 255, after prediction the residuals can assume values from -255 to 255. This is also the same idea in the color indexed videos with a palette of n colors. The indexes to the colormap can assume values from 0 to n-1 and the residuals will be within a range of $-(n-1)$ to $n-1$.

However, it is not possible to construct a table with statistics for negative values. So, it is necessary to add a fixed offset to shift all the negative values. This offset will be 255 for gray scale videos and $n-1$ for color indexed videos with a set of n colors.

Motion vectors can assume a value from 0 to the range of seeking around the neighborhood of the reference block. Once the search can be made in any direction within the seek range, motion vectors can also be negative. Therefore, there are $2\times$ seek range $+1$ possible values. Once again, it is necessary to shift the negative values with an offset and, for the motion vectors, this offset is the seeking range.

The palette is not predicted, it is sent directly to the encoder. Its table of probabilities has 256^{order} entries, which is the number of possible values that each R, G and B component can assume. This table must be of this size because, even though the set of colors have been reduced, those three components have unknown values and they can assume one of 256 possible values, which can originate one of 256^3 possible colors. The statistics of the three components are updated without any distinction, in the same statistics table. Each component of the same palette is encoded one after another and then, another palette entry can be sent to the encoder in order to repeat the process until all the palette entries have been processed.

Chapter 5

Experimental Results

In this chapter, we present experimental results obtained by developed video codecs in this thesis and with the H.264 and JPEG-2000 codecs providing a comparison among them. The video sequences used in our experiments can be found at [3]. In Table 5.1 it is presented the information of the videos. Those videos are in YUV format and therefore, they were adapted whether to gray scale or to color indexed video sequences. Three frames of each video (the first, the middle and the last one) are presented in Annex A, where from Fig. A.1 to Fig. A.7 are from Gray Scale videos, from Fig. A.8 to Fig. A.14 are from Color Indexed videos limited to a palette of 8 colors and from Fig. A.15 to Fig. A.21 are from color indexed videos limited to a palette of 256 colors.

Video	**Resolution**	**Nr. of Frames**	**Frequency(Hz)**
Akyio	QCIF (176×144)	300	30
Bus	CIF (352×288)	150	30
Carphone	QCIF (176×144)	382	30
Claire	QCIF (176×144)	494	30
Coastguard	QCIF (176×144)	300	30
Mobile	QCIF (176×144)	300	30
News	QCIF (176×144)	300	30

Table 5.1: Information of the video sequences.

5.1 Entropy Values

In this section, we present the entropy of the videos used in the experimental results. We will start by the first order entropy for gray scale and videos with a set of 8 and 256 colors and then the entropy of the residuals and the motion vectors.

5.1.1 First Order Entropy

Table 5.2 expresses the mean of the first order entropy of gray scale videos. The variation of the first order entropy along the frames of the video is presented in Fig. 5.1.

As it can be seen, the most complex video to encode is the "Mobile". On the other hand, "Claire" is the less complex to encode.

Video	**Entropy (bits per symbol)**
Akiyo	7.14
Bus	7.16
Carphone	7.19
Claire	6.28
Coastguard	7.30
Mobile	7.68
News	6.99

Table 5.2: Mean of the first order entropy of gray scale videos.

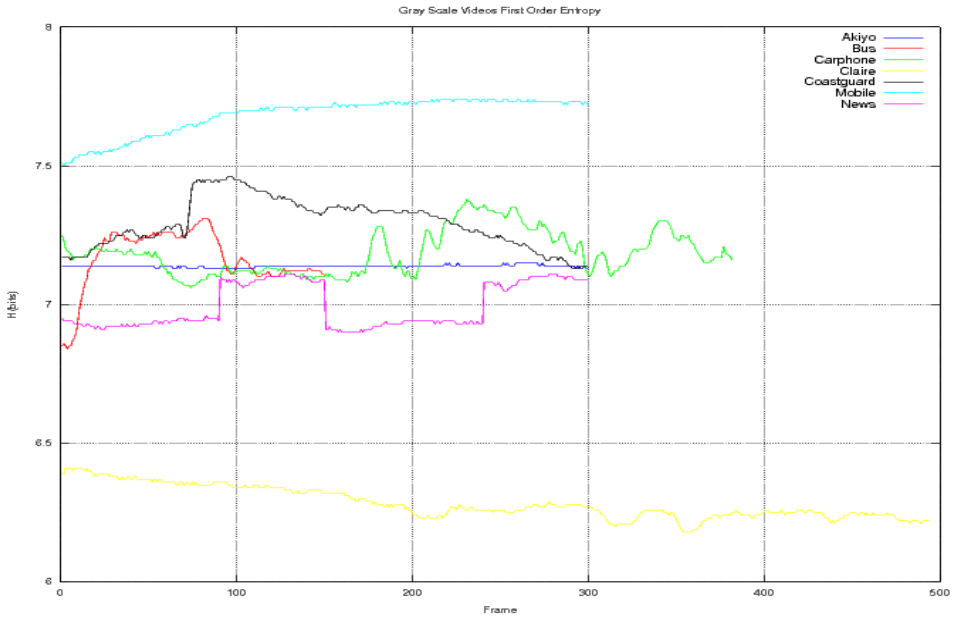

Figure 5.1: First order entropy of the gray scale videos along all their frames.

Table 5.3 expresses the mean of the first order entropy of videos with a set of 8 colors. The first column reflects the entropy of the indexes assigned to each pixel and the second is related to the entropy of the palette. The variation of the first order entropy of the indexes along all the frames of the video is presented in Fig. 5.2.

According to the obtained results, the entropy of the indexes is very low. This is due to the values of the indexes, which vary from 0 to 7. Regarding the colormap entropy, the values are higher than the indexes, which indicates that the palette is more complex to encode than the indexes.

Table 5.4 expresses the mean of the first order entropy of videos with a palette of 256 colors. The first column reflects the entropy of the indexes assigned to each pixel and the second is related to the entropy of the palette. The first order entropy of the indexes for each frame is depicted in the graphic presented in Fig. 5.3 for all the videos used in the experimental results.

Analyzing these results, the entropy has reached higher values for the indexes than the ones obtained for videos with a set of 8 colors. This result was expected, once there are 256 colors, and the less frequent colors are also displayed, which increases the entropy value. Regarding

Video	Entropy (bits per symbol)	Palette Entropy
Akiyo	2.90	4.42
Bus	2.98	4.41
Carphone	2.98	4.50
Claire	2.85	4.39
Coastguard	2.95	4.44
Mobile	2.95	4.53
News	2.94	4.31

Table 5.3: Mean of the first order entropy of color indexed videos with a set of 8 Colors.

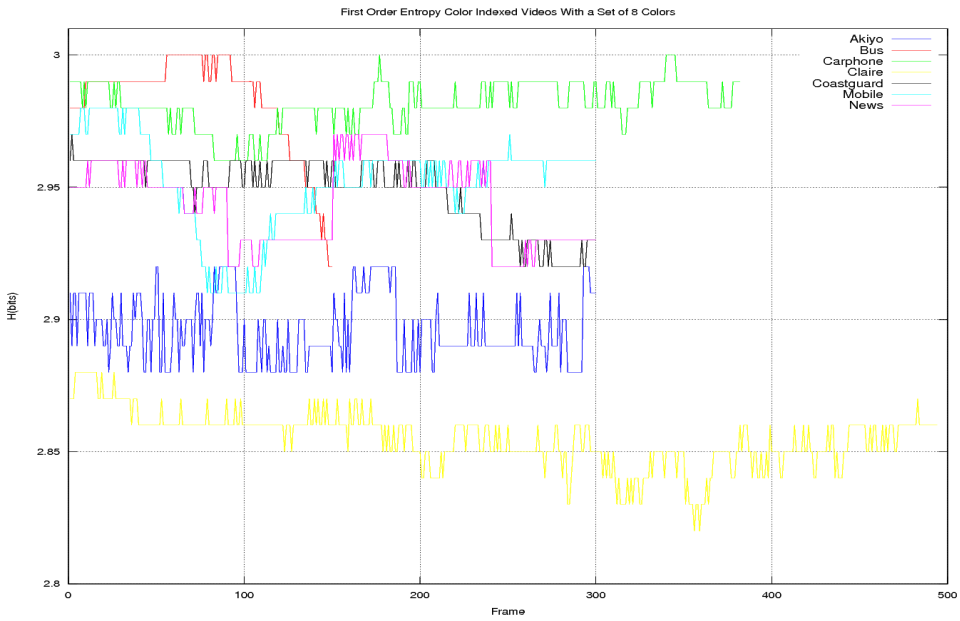

Figure 5.2: First order entropy of videos with a set of 8 colors along frames.

the colormap, the entropy is also high, which shows that achieving great compression results with the palette could be difficult.

Video	Entropy (bits per symbol)	Palette Entropy
Akiyo	6.10	6.87
Bus	7.46	7.15
Carphone	7.11	7.29
Claire	6.48	6.97
Coastguard	7.50	7.33
Mobile	6.19	7.42
News	7.11	7.18

Table 5.4: Mean of the first order entropy of color indexed videos with a set of 256 colors.

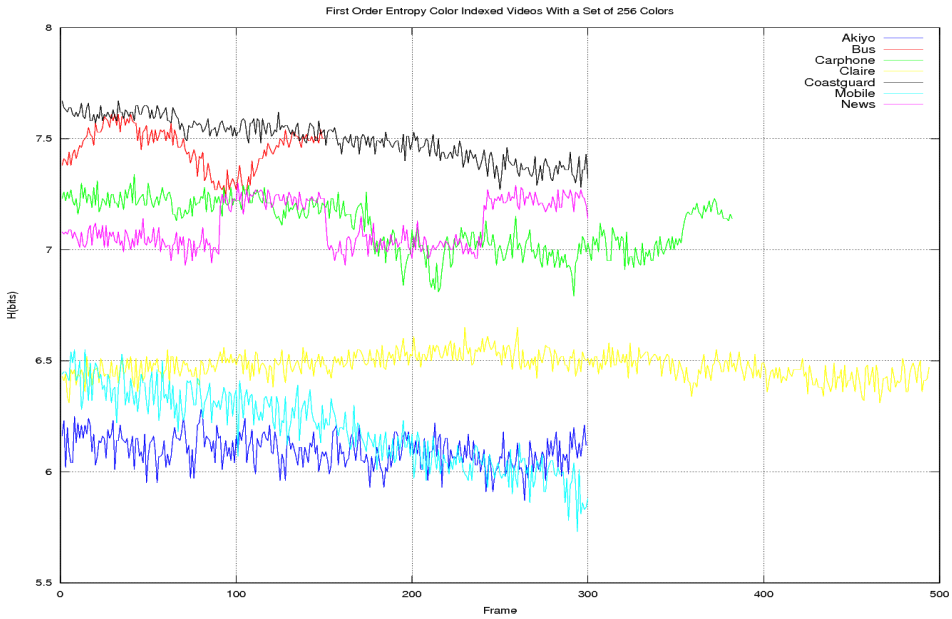

Figure 5.3: First order entropy of videos with a set of 256 colors along frames.

5.1.2 Residuals and Motion Vectors Entropy

As explained in Section 4.3, the video information is predicted before being encoded. Therefore, the values that are sent to the encoder are the residuals, with or without the MED prediction step, and the motion vectors. Hence, to better analyze how much can the file size be reduced, we calculated the entropy of those values for each video.

Table 5.5 shows those values for gray scale videos. The entropy values for the motion vectors are lower than the residuals in all the cases. The use of the MED Predictor does not decrease the entropy of the residuals in most of the times and, when it does, the improvement is not too substantial.

Video	**R. E. (bps)**	**R. E. MED (bps)**	**M. V. E. (bps)**
Akiyo	1.66	1.76	0.66
Bus	4.91	4.59	3.03
Carphone	3.69	3.67	2.41
Claire	2.04	2.25	1.24
Coastguard	4.20	4.17	1.63
Mobile	5.33	5.57	0.95
News	2.4	2.45	0.88

Table 5.5: Mean of the residuals entropy of gray scale videos.
Description: R. E. : Residuals entropy; R. E. MED: Residuals entropy after using the MED predictor; M. V. E. : Motion vectors entropy; bps: bits per symbol

---

It is well known that for color-indexed videos the palette reordering can also improve the efficiency of the encoder. Therefore, we calculated the results for *pre* and *post* palette-reordered videos.

Tables 5.6 and 5.7 show the results for videos with a colormap of 8 colors.

Here, the entropy values for the residuals are very low and, once again, MED Predictor does not bring any improvement. Although it was expected to have some improvements in the entropy after reordering the palette, the results in *post* palette-reordering videos were even lower than the expected. Color-indexed videos with a set of 8 colors can achieve great compression rates.

Video	**R. E. (bps)**	**R. E. MED (bps)**	**M. V. E. (bps)**
Akiyo	0.32	0.47	2.45
Bus	2.37	2.52	4.38
Carphone	1.08	1.35	3.47
Claire	0.79	0.87	3.49
Coastguard	1.58	1.84	3.12
Mobile	2.45	2.72	3.63
News	0.60	0.83	2.15

Table 5.6: Mean of the residuals entropy of color indexed videos with a set of 8 colors.
Description: R. E. : Residuals entropy; R. E. MED: Residuals entropy after using the MED predictor; M. V. E. : Motion vectors entropy; bps: bits per symbol

Video	**R. E. (bps)**	**R. E. MED (bps)**	**M. V. E. (bps)**
Akiyo	0.27	0.42	2.40
Bus	1.34	1.57	3.27
Carphone	0.63	0.91	3.24
Claire	0.28	0.49	3.34
Coastguard	0.90	1.24	1.92
Mobile	1.24	1.70	1.36
News	0.49	0.63	2.09

Table 5.7: Mean of the residuals entropy of color indexed videos with a set of 8 colors after being palette reordering.
Description: R. E. : Residuals entropy; R. E. MED: Residuals entropy after using the MED predictor; M. V. E. : Motion vectors entropy; bps: bits per symbol

Analyzing the results for the videos with 256 colors (Tables 5.8 and 5.9), reordering the palette reduced the entropy substantially. In these videos, using the MED predictor sometimes decreased the entropy. However, that improvement was not significant.

Video	**R. E. (bps)**	**R. E. MED (bps)**	**M. V. E. (bps)**
Akiyo	4.98	4.60	3.04
Bus	7.59	8.02	4.83
Carphone	6.71	6.74	4.37
Claire	5.18	4.83	3.89
Coastguard	7.22	7.78	4.37
Mobile	6.74	6.93	4.31
News	6.08	6.10	2.99

Table 5.8: Mean of the residuals entropy of color indexed videos with 256 colors.
Description: R. E. : Residuals entropy; R. E. MED: Residuals entropy after being predicted with MED; M. V. E. : Motion vectors entropy; bps: bits per symbol

Video	**R. E. (bps)**	**R. E. MED (bps)**	**M. V. E. (bps)**
Akiyo	3.95	3.50	2.33
Bus	5.91	5.85	3.11
Carphone	4.74	4.79	2.91
Claire	4.24	4.10	2.84
Coastguard	5.22	5.39	1.65
Mobile	5.59	5.86	1.27
News	4.38	4.32	1.54

Table 5.9: Mean of the residuals entropy of color indexed videos with a set of 256 colors after being reordered the palette.
Description: R. E. : Residuals entropy; R. E. MED: Residuals entropy after being predicted with MED; M. V. E. : Motion vectors entropy; bps: bits per symbol

5.1.3 Overall Results

Observing these results, it can be deduced that the files that are going to accomplish a higher compression rate will be the color indexed videos with a set of 8 colors, which is due to the indexes of the palette assume values only from 0 to 7. On the other hand, videos with a palette of 256 colors seems to be the ones which will not achieve a big compression rate. However, one thing is consensual: the palette, whether is 8 or 256, has a significant entropy and it is not easy to encode it. It was due to this reason that the strategy for the algorithm using the Golomb codes was to send straight to the bitstream the palette without encoding it.

Most of the gray scale videos have also a great redundancy to exploit and hence, good results can be accomplished, using a proper encoding method.

It can also be implied from these results that reordering the palette of color indexed videos can bring an improvement in the final size of the video, specially for the 256 colors videos, with a slight decreasing in the entropy values.

5.2 Encoding results with Golomb codes

In this section we present encoding results with the Golomb codes. The encoding process was made according with the information obtained by the entropy values depicted from Table 5.5 to 5.9, to decide which was the best number of bits to encode the residuals and the motion vectors, as well as if it was better to use the MED predictor or not. The values of the block size and the seeking range were always set as 8×8 and 16×16, respectively.

The encoding results for gray scale videos, videos with a set of 8 and 256 colors are detailed in Tables 5.10, 5.11 and 5.12, respectively.

Looking into these results, the objective of reducing the original size of the video has been accomplished. However, it was expected values closer to the entropy, specially for color indexed videos with a colormap of 8 colors. These results are due to the fact that the palette is a difficult source of data to explore some redundancy. Nevertheless, encoding the palette would not be the solution, on the contrary, it would increase the size of the compressed file.

About videos with the palette reordered, the compression results were better, specially for videos with 256 colors, as expected.

Video	**Original size** (bits)	**Compressed Size**	
		(bits)	(bpp)
Akiyo	60 825 704	25 337 104	3.33
Bus	121 651 304	75 895 760	4.99
Carphone	77 451 368	41 662 112	4.30
Claire	100 159 600	41 707 984	3.33
Coastguard	60 825 704	35 103 912	4.62
Mobile	60 825 704	44 031 952	5.79
News	60 825 704	29 145 232	3.83

Table 5.10: Encoding results with the Golomb codes for gray scale videos.

Video	**Original size** (bits)	**Compressed Size**			
		***Pre* Palette Reordering**		***Post* Palette Reordering**	
		(bits)	(bpp)	(bits)	(bpp)
Akiyo	60 883 304	24 097 904	3.17	24 015 512	3.16
Bus	121 680 104	52 510 112	3.45	48 448 336	3.19
Carphone	77 524 712	31 537 352	3.26	30 525 496	3.15
Claire	100 254 440	40 230 152	3.21	40 035 336	3.20
Coastguard	60 883 304	24 795 128	3.26	23 799 960	3.13
Mobile	60 883 304	26 605 928	3.50	24 135 112	3.17
News	60 883 304	24 142 512	3.18	23 885 928	3.14

Table 5.11: Encoding results with the Golomb codes for videos with a palette of 8 colors.

Video	**Original size** (bits)	**Compressed Size**			
		***Pre* Palette Reordering**		***Post* Palette Reordering**	
		(bits)	(bpp)	(bits)	(bpp)
Akiyo	62 668 920	48 514 016	6.38	37 579 968	4.95
Bus	122 572 920	121 753 448	8.01	100 698 016	6.62
Carphone	79 798 392	72 757 744	7.52	52 732 840	5.45
Claire	103 194 744	80 988 400	6.47	64 779 648	5.17
Coastguard	62 668 920	59 957 632	7.89	45 213 312	5.99
Mobile	62 668 920	58 143 344	7.65	49 484 504	6.51
News	62 668 920	55 849 768	7.35	40 297 840	5.30

Table 5.12: Encoding results with the Golomb codes for videos with a palette of 256 colors.

5.3 Encoding results with Arithmetic coding

In this section we present the results obtained with our encoder based on arithmetic coding. With this encoding method, it was used for each video three three finite-context model orders: 0, 1 and 2. The size of the reference block was set to 8×8 and the size of the seek block to 16×16.

The encoding results for gray scale videos, videos with a set of 8 and 256 colors are detailed in Tables 5.13, 5.14 and 5.15, respectively.

Video	**Original size** (bits)	**Order**	**Compressed Size**	
			(bits)	(bpp)
Akiyo	60 825 704	0	13 940 968	1.83
		1	23 874 592	3.14
		2	31 865 576	4.19
Bus	121 651 304	0	76 759 144	5.05
		1	143 250 448	9.42
		2	191 516 560	12.59
Carphone	77 451 368	0	37 724 384	3.90
		1	68 160 688	7.04
		2	91 521 072	9.45
Claire	100 159 600	0	27 895 128	2.23
		1	50 125 392	4.01
		2	69 820 792	5.58
Coastguard	60 825 704	0	33 367 136	4.39
		1	60 794 024	7.99
		2	81 191 872	10.68
Mobile	60 825 704	0	41 735 720	5.49
		1	75 709 712	9.96
		2	93 001 544	12.23
News	60 825 704	0	19 523 504	2.57
		1	33 353 872	4.39
		2	43 768 944	5.76

Table 5.13: Encoding results for gray scale videos.

Video	**Original size**	**Order**	**Compressed Size**			
			***Pre* Palette Reord.**		***Post* Palette Reord.**	
	(bits)		(bits)	(bpp)	(bits)	(bpp)
Akiyo	60 883 304	0	3 134 000	0.41	2 768 488	0.36
		1	5 604 176	0.74	5 017 832	0.66
		2	7 822 584	1.03	7 046 544	0.93
Bus	121 680 104	0	38 143 624	2.51	22 112 232	1.45
		1	72 115 096	4.74	42 214 960	2.78
		2	103 721 296	6.82	61 182 224	4.02
Carphone	77 524 712	0	11 664 904	1.20	7 230 160	0.75
		1	20 672 600	2.14	13 423 592	1.39
		2	28 743 896	2.97	19 144 424	1.98
Claire	100 254 440	0	11 519 056	0.92	5 022 504	0.40
		1	18 310 160	1.46	9 149 480	0.73
		2	24 135 512	1.93	12 702 552	1.01
Coastguard	60 883 304	0	12 861 848	1.69	7 459 464	0.98
		1	23 216 072	3.05	13 974 040	1.84
		2	32 856 064	4.32	20 240 280	2.66
Mobile	60 883 304	0	19 567 888	2.57	9 886 208	1.30
		1	36 007 768	4.74	19 089 192	2.51
		2	50 798 776	6.68	27 904 784	3.67
News	60 883 304	0	5 198 400	0.68	4 362 240	0.57
		1	9 484 264	1.25	8 026 024	1.06
		2	13 350 224	1.76	11 418 592	1.50

Table 5.14: Encoding results for videos with a set of 8 colors.

Video	**Original size**	**Order**	**Compressed Size**			
			***Pre* Palette Reord.**		***Post* Palette Reord.**	
	(bits)		(bits)	(bpp)	(bits)	(bpp)
Akiyo	62 668 920	0	41 222 304	5.42	33 311 944	4.38
		1	66 751 928	8.78	55 699 632	7.33
		2	80 727 064	10.62	72 670 752	9.56
Bus	122 572 920	0	118 946 072	7.82	92 761 672	6.10
		1	212 521 408	13.98	171 977 344	11.31
		2	233 511 872	15.36	21 6993 632	14.27
Carphone	79 798 392	0	69 553 120	7.18	50 335 864	5.20
		1	112 155 184	11.59	88 506 992	9.14
		2	125 048 152	12.92	113 532 040	11.73
Claire	103 194 744	0	70 731 664	5.65	58 730 680	4.69
		1	117 344 248	9.37	101 877 008	8.13
		2	142 065 632	11.35	131 610 080	10.51
Coastguard	62 668 920	0	58 462 184	7.69	42 793 488	5.63
		1	97 313 352	12.80	76 191 272	10.02
		2	104 334 824	13.72	95 565 360	12.57
Mobile	62 668 920	0	54 847 296	7.21	45 547 208	5.99
		1	87 755 208	11.54	76 616 352	10.08
		2	96 754 672	12.73	90 691 120	11.93
News	62 668 920	0	49 420 168	6.50	36 430 904	4.79
		1	79 028 560	10.39	62 793 200	8.26
		2	90 899 784	11.96	80 914 032	10.64

Table 5.15: Encoding results for videos with a set of 256 colors.

Analyzing the results, we can see that using a order-1 or 2 finite-context model does not bring any improvement to the compression rate. Moreover, in some cases, the compressed files are bigger than the original one, like with the "Bus" or the "Mobile" video for gray scale format and for videos with a palette of 256 colors. With videos with a set of 8 colors, it did not happen, once the entropy values were low. However, the increase tendency of the compression rate with the increasing of the order did happen anyway.

This developed algorithm is extremely efficient for color indexed videos with a set of 8 colors and, for gray scale and videos with 256 colors, it is also efficient when the entropy of the data is low.

5.4 Encoding results with JPEG-2000 standard

In this section we present the results of the encoding process when it was applied the JPEG 2000 codec. It was used an open source program provided by the Jasper project [43], which offers a software based on Part 1 of the standard. The used version was the 1.900.1.

As it was detailed on Section 2.6.3, this codec is applied only to images. So, in order to perform an encoding process to all the videos, it was necessary to split all the frames. For color indexed videos, the frames used were the ones produced after being applied the *ppmquant* command to quantize them, which was one of the middle steps to transform a color video into a color indexed one, as it was described in Section 4.2.2. In the case of gray scale videos, the same program developed to split all the frames (B.5) of color videos had an option, which allows to split gray scale videos. These split frames were transformed into PGM files.

After having all the frames split and converted into images, compressing all the frames is the next step. Another shell script (Annex B.16.2) was developed to compress all the frames of the same video in just one command call, once if it was done manually, it would take too much time. The compression results are the sum of all frames of the video, and they are presented in Table 5.16, 5.17 and 5.18, for gray scale, color indexed with a set of 8 and 256 colors, respectively.

Video	**Original size** (bits)	**Compressed Size** (bits)	(bpp)
Akiyo	60 825 704	27 188 040	3.58
Bus	121 651 304	74 426 520	4.89
Carphone	77 451 368	40 444 600	4.18
Claire	100 159 600	35 689 504	2.85
Coastguard	60 825 704	39 158 384	5.15
Mobile	60 825 704	48 565 224	6.38
News	60 825 704	32 460 048	4.27

Table 5.16: Encoding results for gray scale videos.

The results obtained for gray scale videos are good, being quite similar to the ones achieved with the two proposed encoding algorithms. However, they were always higher on almost cases, except in the results of the "Bus" video.

Video	**Original size** (bits)	**Compressed Size** (bits)	(bpp)
Akiyo	182 611 104	59 078 632	7.77
Bus	365 004 368	158 472 456	10.42
Carphone	232 563 472	93 306 552	9.64
Claire	300 701 584	65 101 872	5.20
Coastguard	182 611 104	76 800 144	10.10
Mobile	182 676 640	133 951 880	17.62
News	182 611 104	72 267 472	9.50

Table 5.17: Encoding results for videos with a set of 8 colors.

The results shown in Table 5.17 are good, taking into account that the files sent to encode were composed by 24 bits per pixel. All the compression rates were good, the only exception was with the Mobile video, which had a bad performance.

Video	Original size (bits)	Compressed Size (bits)	(bpp)
Akiyo	182 611 104	76 795 048	10.10
Bus	365 004 368	173 123 592	11.38
Carphone	232 563 472	108 287 688	11.19
Claire	300 701 584	110 309 344	8.81
Coastguard	182 611 104	85 505 008	11.25
Mobile	182 676 640	140 536 992	18.48
News	182 611 104	88 345 768	11.62

Table 5.18: Encoding results for videos with a set of 256 colors.

The results shown in Table 5.18 indicates that this standard has a reasonable performance. However, these results are higher than the ones obtained with the two proposed algorithms in this thesis, which attains the objective of this work.

5.5 Encoding results with H.264/AVC standard

The results of the encoding process with the H.264/AVC standard in a lossless mode are depicted in this section. The software used in this thesis to encode with this standard was provided by JVT [44]. We used the version 14.2. Once this standard is not capable of receiving a single information plane, as proposed during this thesis, all the gray scale videos files have been converted to the YUV color-space and for the RGB color-space, when the videos have a reduced set of colors.

If the encoding process is focused on gray scale videos, the input file that is sent to the encoder is the original YUV file found at [3]. This standard has an option that, when enabled, will send to the encoder all the chrominance information set to zero.

When dealing with files with a reduced set of colors, a program has been developed (Annex B.8) to receive all the frames that have been quantized with the *ppmquant* command and, instead of building a palette and assigning an index to each pixel, it will assemble all the frames altogether in order to construct a RGB file which can be accepted as a input file for the H.264 standard to encode it.

The following tables show the obtained results. In Table 5.19 it is presented the encoding results of the gray scale videos and in Table 5.20 and 5.20 are depicted the results when encoding videos with a reduced set of 8 and 256 colors, respectively.

Video	**Original size** (bits)	**Compressed Size** (bits)	(bpp)
Akiyo	182 476 800	6 813 904	0.90
Bus	364 953 600	61 248 768	4.03
Carphone	232 353 792	29 581 376	3.06
Claire	300 478 464	20 744 264	1.66
Coastguard	182 476 800	27 102 824	3.56
Mobile	182 476 800	29 349 104	3.86
News	182 476 800	11 332 304	1.49

Table 5.19: Encoding results for gray scale videos.

Analyzing Table 5.19, the results are very satisfactory. Taking into account that each pixel of the original video has a 24 bit per pixel representation, and after encoding, having been obtained such high compression rate, this standard shows that performs very well with these kind of videos.

Video	**Original size** (bits)	**Compressed Size** (bits)	(bpp)
Akiyo	182 476 800	12 875 640	1.69
Bus	364 953 600	203 383 456	13.38
Carphone	232 353 792	63 500 720	6.56
Claire	300 478 464	26 565 424	2.12
Coastguard	182 476 800	74 190 736	9.76
Mobile	182 476 800	104 582 840	13.76
News	182 476 800	23 071 400	3.03

Table 5.20: Encoding results for videos with a set of 8 colors.

Looking into the results shown in Table 5.20, it is notorious that H.264/AVC has also a good performance with this kind of videos. However, when dealing with videos which have higher entropy values, the compression rates are not as good as when dealing with the same videos but in a gray scale format.

The results in Table 5.21 show that the performance for videos with high entropy values is a little poor, specially when looking to the result of the “Mobile” video. Videos with low entropy values are within the range of the results obtained with the proposed algorithms in this thesis, but a somewhat higher.

Video	Original size (bits)	Compressed Size (bits)	(bpp)
Akiyo	182 476 800	51 466 568	6.77
Bus	364 953 600	206 602 416	13.59
Carphone	232 353 792	103 294 144	10.67
Claire	300 478 464	85 475 056	6.82
Coastguard	182 476 800	94 496 792	12.43
Mobile	182 476 800	144 359 640	18.99
News	182 476 800	62 930 240	8.27

Table 5.21: Encoding results for videos with a set of 256 colors.

5.6 Comparing the results

In this section we discuss the results obtained by the two proposed methods based on a hybrid model, one with the Golomb codes and the other with arithmetic coding, and the standard used for images (JPEG2000), and by the one used for videos (H.264/AVC).

Starting by comparing the two proposed models in this thesis, the performance of the method based on arithmetic coding was always better, except for the "Bus" video in gray scale format. This better performance was even more notorious for videos with a reduced set of 8 colors, where arithmetic coding has shown to be, by far, the best choice to encode these type of videos.

Comparing the results of the two proposed encoders with the JPEG-2000 standard, the proposed methods were better in almost all videos. Exceptions happened when encoding gray scale videos, where JPEG-2000 had a better performance than the hybrid model with the Golomb codes with "Bus", "Carphone" and "Claire" videos. JPEG-2000 was only better than the arithmetic coding when encoding the "Bus" video, also in a gray scale format.

The results of the H.264/AVC standard for gray scale videos were outstanding. Its great performance was even better than the ones achieved by the arithmetic coding. These results indicate that the proposed encoders have to be improved, in order to be better efficient to use a single information plane to represent gray scale videos than YUV. For color-indexed videos, the results were better when encoding with the arithmetic coding, which demonstrate that representing videos with a small set of colors having an indexed palette with those colors might be a valid option, as it brings better compression rates.

Chapter 6

Conclusions and future work

An approach to represent videos with a single information plane have been studied in this thesis, as well as two proposed encoding algorithms. Analyzing and comparing these two developed algorithms, the one that has shown to be most effective, whether for gray scale or color-indexed videos, has been arithmetic coding. Even though the hybrid model using the Golomb codes has shown a good performance, the option of using the MED predictor rarely brought improvements to the algorithm. As future work, another predictor can be developed in order to see if can be more efficient than the MED predictor.

As mentioned above, arithmetic coding was the best of the two developed encoding methods. However, the results when using a 1 or 2-order finite context model were below average and, in some cases, the size of the compressed file was higher than the original one. Furthermore, its high performance has happened when using a 0-order finite context model. This issue should also be referred in a future work.

Comparing those results with the ones obtained when using well known standards, such as JPEG-2000 and H.264/AVC, the results were better in most of the cases. It has been notorious that, for gray scale files, JPEG-2000 sometimes was better and, by far, H.264/AVC has demonstrated that it has a good performance when encoding gray scale videos in a lossless mode.

Therefore, the proposed model of characterizing video information with a single information plane is very effective for videos with a reduced set of colors, specially when they are used a small number of colors, such as 8 colors. Hence, this approach can be applied for web applications when the quality of the videos does not need to be that demanding. On the other hand, representing gray scale videos with this model has not shown to be the best choice, once H.264/AVC still has a better performance. As future work, in order to use this

model for medical information, such as CT Scans or sonography, another approaches for the encoding algorithms can be studied, and maybe, trying to adapt the H.264/AVC standard to receive videos with a single information plane, which could improve even more the results obtained.

Appendix A

Video test sets

This Appendix provides three frames of the videos used to evaluate the performance of the video codecs presented in this thesis, namely the first, the middle and the last frames.

Figure A.1: Frames of the video Akiyo

Figure A.2: Frames of the video Bus

Figure A.3: Frames of the video Carphone

Figure A.4: Frames of the video Claire

Figure A.5: Frames of the video Coastguard

Figure A.6: Frames of the video Mobile

Figure A.7: Frames of the video News

Figure A.8: Frames of the video Akiyo with a set of 8 colors.

Figure A.9: Frames of the video Bus with a set of 8 colors.

Figure A.10: Frames of the video Carphone with a set of 8 colors.

Figure A.11: Frames of the video Claire with a set of 8 colors.

Figure A.12: Frames of the video Coastguard with a set of 8 colors.

Figure A.13: Frames of the video Mobile with a set of 8 colors.

Figure A.14: Frames of the video News with a set of 8 colors.

Figure A.15: Frames of the video Akiyo with a set of 256 colors.

Figure A.16: Frames of the video Bus with a set of 256 colors.

Figure A.17: Frames of the video Carphone with a set of 256 colors.

Figure A.18: Frames of the video Claire with a set of 256 colors.

Figure A.19: Frames of the video Coastguard with a set of 256 colors.

Figure A.20: Frames of the video Mobile with a set of 256 colors.

Figure A.21: Frames of the video News with a set of 256 colors.

Appendix B

Video Tools

B.1 Data Structure

The data structures used in this thesis are the following ones:

- OnePlanFrame (this structure represents the video information characterized by the one plane proposed model);
- RGBFrame (this is a structure that represents the video information in a RGB mode. Used specially in some algorithms to adpat that information into One Plane Information videos);
- Yuv420Frame (this is a structure that represents the video information in a YUV 4:2:0 mode. Used specially in some algorithms to adpat that information into One Plane Information videos);
- Yuv444Frame (this is a structure that represents the video information in a YUV 4:4:4 mode. Used specially in some algorithms to adpat that information into One Plane Information videos);

B.2 File List

Here is a list of all the documented files which provide the basic functions used in the developed programs in this thesis:

- conversions.c (This module handles the basic operation of video sequences);

- conversions.h (This is the header file for the basic conversions of video sequences and another helpful functions);
- rgb.c (This module handles the basic operations on RGB video sequences);
- rgb.h (This is header file for the basic operations on RGB video sequences);
- yuv420.c (This module handles the basic operations on YUV 4:2:0 video sequences);
- yuv420.h (This is header file for the basic operations on YUV 4:2:0 video sequences);
- yuv444.c (This module handles the basic operations on YUV 4:4:4 video sequences);
- yuv444.h (This is header file for the basic operations on YUV 4:4:4 video sequences);
- one_plan.c (This module handles the basic operations on One Information Plane video sequences);
- one_plan.h (This is header file for the basic operations on One Information Plane video sequences);
- golomb.c (This module handles the basic operations to encode with the Golomb method);
- golomb.h (This is header file for the basic operations to encode with the Golomb method);
- arith_aux.c (This module handles the basic operations to encode with the Arithmetic coding);
- arith_aux.h (This is header file for the basic operations to encode with the Arithmetic coding);
- arithmetic.c (This module handles the basic operations to create the table of probabilities and select the index according with finite-context model);
- arithmetic.h (This is header file for the basic operations to create the table of probabilities and select the index according with finite-context model);
- block.c (This module handles the basic operations of the motion prediction);
- block.h (This is header file for the basic operations of the motion prediction);
- bits.c (This module handles the basic operations of the Golomb codes bitstream read/write);

- bits.h (This is header file for the basic operations of the Golomb codes bitstream read/write);
- bitio.c (This module handles the basic operations of the Arithmetic coding bitstream read/write);
- bitio.h (This is header file for the basic operations of the Arithmetic coding bitstream read/write);

All the developed programs used in thesis are described below:

- VideoCompare.c (This application compares two One Information Plane video sequences and prints the PSNR and RMSE resulting of the comparison process);
- YuvConv.c (This application converts YUV or RGB video sequences into One Information Plane videos in gray scale format);
- YuvSplitFrames.c (This application splits all the video sequences (in RGB, YUV 4:2:0, YUV 4:4:4 and One Information Plane) and transform them into PPM or PGM files, if the video is gray scale);
- YuvJoinFrames.c (This application assembles all the PPM files after having been quantized into an One Information Plane video file with a set of n colors);
- PaletteReordering.c (This program applies palette reordering to color indexed videos according to the Luminance);
- YuvJoinFramesRGB.c (This application assembles all the PPM files after having been quantized into a RGB video file with a set of n colors);
- ShowOneInfPlan.c (This program plays One Information Plane video sequences (using SDL));
- EntropyCalculator.c (This application calculates the entropy values for the residuals, the motion vectors and the colormap);
- Entropy1Calculator.c (This application calculates the first order entropy of a video sequence);
- BlockEnc.c (This program applies the Golomb codes into a video sequence and send the encoded values into a bitstream);

- BlockDec.c(This program receives the encoded bitstream and performs the decoding algorithm according with the Golomb method);
- BlockEncArith.c (This program applies the Arithmetic coding into a video sequence and send the encoded values into a bitstream);
- BlockDecArith.c(This program receives the encoded bitstream and performs the decoding algorithm according with the Arithmetic coding);

B.3 VideoCompare.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "rgb.h"
#include "conversions.h"
```

Author: Arturo Rodrigues

Version: 1.2

Date: Creation: 27/01/2009

Last Modification: 27/02/2009

B.3.1 Function Documentation

int main (int *argc*, char * *argv*[ ])

Main function

Usage:

./**VideoCompare** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-fh (If video contains file header)]

[-h (Videos Height)]

[-SIF] [-CIF] [-QCIF]

Base video sizes

[-gray]

[-index]

[-rgb]

NOTES:

∗ FileName1 in Gray Scale or Color Indexed Format

∗ FileName2 in RGB, YUV444, Gray Scale or Color Indexed Format

∗ QCIF (177x144)

∗ CIF (352x288)

∗ SIF (NTSC) (177x144)

B.4 YuvConv.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "yuv420.h"
#include "yuv444.h"
#include "one_plan.h"
#include "conversions.h"
```

Enumerations

typedef enum {

YUV420_ONEPLAN,

YUV444_ONEPLAN,

EMPTY

} T_CONV;

Author: Arturo Rodrigues

Version: 1.2

Date: Creation: 27/01/2009

Last Modification: 27/02/2009

B.4.1 Function Documentation

int main (int *argc*, char * *argv*[ ])

Main function

Usage:

./**YuvConv** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-SIF] [-CIF] [-QCIF]

* Base video sizes

[-fh (If the YUV file has header with w and h)]

[-420 (YUV420 format to Gray Scale)]

[-444 (YUV444 format to Gray Scale)]

NOTES:

* FileName1 YUV format

* FileName2 the new file in Gray Scale

* QCIF (177x144)

* CIF (352x288)

* SIF (NTSC) (177x144)

B.5 YuvSplitFrames.c File Reference

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "yuv420.h"
#include "yuv444.h"
#include "rgb.h"
#include "conversions.h"
```

Author: Arturo Rodrigues

Version: 1.2

Date: Creation: 09/03/2009

Last Modification: 17/03/2009

B.5.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**YuvSplitFrames** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-SIF] [-CIF] [-QCIF]

* Base video sizes

[-fh (If the YUV file has header with w and h)]

[-420 (Video format)]

[-444 (Video format)]

[-rgb (Video format)]

NOTES:

* FileName1 RGB or YUV format
* FileName2 is the picture in PPM format
* QCIF (177x144)
* CIF (352x288)
* SIF (NTSC) (177x144)

B.6 YuvJoinFrames.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "rgb.h"
#include "conversions.h"
```

Author: Arturo Rodrigues

Version: 1.1

Date: Creation: 12/03/2009

Last Modification: 17/03/2009

B.6.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**YuvJoinFrames** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-fn (Number of frames to join)]

[-nr (number of colors of the PPM)]

NOTES:

∗ FileName1 in PPM format

∗ FileName2 is the name of the video

B.7 PaletteReordering.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "one_plan.h"
```

Author: Arturo Rodrigues

Version: 1.0

Date: Creation: 19/04/2009

Last Modification: 19/04/2009

B.7.1 Function Documentation

int main (int *argc*, char ∗ *argv*[ ])

Main function

Usage:

./**PaletteReordering FileName1 Filename2**

NOTES:

∗ FileName1 and FileName2 are One Information Plane videos

B.8 YuvJoinFramesRGB.c File Reference

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "rgb.h"
#include "conversions.h"
```

Author: Arturo Rodrigues

Version: 1.0

Date: Creation: 14/09/2009

Last Modification: 14/09/2009

B.8.1 Function Documentation

int main (int *argc*, char * *argv*[ ])

Main function

Usage:

./**YuvJoinFramesRGB** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-fn (Number of frames to join)]

NOTES:

* FileName1 in PPM format

* FileName2 is the name of the output RGB file

B.9 ShowOneInfPlan.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL.h>
```

```
#include "one_plan.h"
#include "conversions.h"
```

Author: Arturo Rodrigues

Version: 1.1

Date: Creation: 12/02/2009

Last Modification: 27/02/2009

B.9.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**ShowOneInfPlan** [**OPTIONS**] **FileName**

OPTIONS:

[-SIF] [-CIF] [-QCIF]

* Base video sizes

[-fh (If the file has header with w and h)]

[-nr (The number of colors of the video)]

[-wait (stop after first frame)]

[-2 (display in double size)]

[-v (verbose)]

NOTES:

* FileName is the One Information Plane video

* QCIF (177x144)

* CIF (352x288)

* SIF (NTSC) (177x144)

B.10 EntropyCalculator.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "conversions.h"
#include "golomb.h"
```

Author: Arturo Rodrigues

Version: 1.0

Date: Creation: 27/07/2009

Last Modification: 27/07/2009

B.10.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**EntropyCalculator** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-SIF] [-CIF] [-QCIF]

* Base video sizes

[-fh (If video contains file header)]

[-clr If video is color indexed(def. 0)]

[-bs blockSize(def. 8×8)]

[-seek seekSize(def. 16×16)]

NOTES:

* FileName1 in One Plane format
* FileName2 the file with Entropy Data
* QCIF (177x144)
* CIF (352x288)
* SIF (NTSC) (177x144)

B.11 Entropy1Calculator.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "conversions.h"
#include "golomb.h"
```

Author: Arturo Rodrigues

Version: 1.1

Date: Creation: 12/06/2009

Last Modification: 27/07/2009

B.11.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**Entropy1Calculator** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-SIF] [-CIF] [-QCIF]

∗ Base video sizes

[-fh (If video contains file header)]

[-clr If video is color indexed(def. 0)]

NOTES:

∗ FileName1 in One Plane format

∗ FileName2 the file with Entropy Data

∗ QCIF (177x144)

∗ CIF (352x288)

∗ SIF (NTSC) (177x144)

B.12 BlockEnc.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "bits.h"
#include "golomb.h"
#include "block.h"
#define START_OF_FRAME 0xffff
```

Author: Arturo Rodrigues

Version: 1.1

Date: Creation: 12/03/2009

Last Modification: 27/03/2009

B.12.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**BlockEnc** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-b nBits (def.4)]

[-vector nBitsVector(def. 1)]

[-bs blockSize(def. 8 × 8)]

[-seek seekSize(def. 16 × 16)]

[-med (enables MED predictor)]

NOTES:

* FileName1 the One Information Plane video

* FileName2 the encoded file

B.13 BlockDec.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "bits.h"
#include "golomb.h"
#include "block.h"
#define START_OF_FRAME 0xffff
```

Author: Arturo Rodrigues

Version: 1.1

Date: Creation: 12/03/2009

Last Modification: 27/03/2009

B.13.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**BlockDec FileName1 FileName2**

NOTES:

* FileName1 the encoded file

* FileName2 the decoded One Information Plane video

B.14 BlockEncArith.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "arith.h"
#include "arith_aux.h"
#include "arithmetic.h"
#include "bitio.h"
#include "block.h"
```

Author: Arturo Rodrigues

Version: 1.4

Date: Creation: 12/04/2009

Last Modification: 02/08/2009

B.14.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**BlockEncArith** [**OPTIONS**] **FileName1 FileName2**

OPTIONS:

[-o (Finite Context Model order)]

[-bs blockSize(def. 8 × 8)]

[-seek seekSize(def. 16 × 16)]

NOTES:

* FileName1 the One Information Plane video

* FileName2 the encoded file

B.15 BlockDecArith.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "one_plan.h"
#include "arith.h"
#include "arith_aux.h"
#include "arithmetic.h"
#include "bitio.h"
#include "block.h"
```

Author: Arturo Rodrigues

Version: 1.3

Date: Creation: 12/04/2009

Last Modification: 02/08/2009

B.15.1 Function Documentation

int main (int *argc*, char * *argv*[])

Main function

Usage:

./**BlockDecArith FileName1 FileName2**

NOTES:

* FileName1 the encoded file

* FileName2 the One Information Plane decoded video

B.16 Developed shell scripts

We developed two shell scripts during this thesis: one to call the *ppmquant* command as many times as the number of frames of the video to quantize and other with the same idea but using the command to encode the images with the JPEG-2000 standard. Their code is depicted on the following subsections.

B.16.1 *Ppmquant* shell script

```
#!/bin/bash
echo ‘‘How many colors to the palette?’’
read valor
echo ‘‘What is the name of the output file?’’
read name
echo ‘‘What is the name of the folder?’’
read folder
```

```
mkdir ../$folder
ls
for i in *
do
ppmquant $valor $i > ../$folder/$name$i
done
```

B.16.2 Jasper shell script

```
#!/bin/bash
echo ‘‘What is the name for the output folder?’’
read folder2
echo ‘‘What is the name of the output file?’’
read name
x = 0
mkdir ../$folder2
ls
for i in *
do
x = $((x+1))
jasper −f $i −F ../$folder2/$name$x.jp2 −T jp2
done
```

Bibliography

[1] http://www.institutoforlanini.com.br/teste/images/ecografia.jpg

[2] http://commons.wikimedia.org/wiki/File:Head_CT_scan.jpg

[3] http://trace.eas.asu.edu/yuv/index.html

[4] A. J. Pinho. Codificação de Áudio e Vídeo. http://www.ieeta.pt/˜ ap/cav .

[5] E. Bodden, M. Clasen and J. Kneis. Arithmetic Coding revealed - A guided tour from theory to praxis. Technical report, School of Computer Science, Sable Research Group, May 25,2007. McGill University.

[6] MPEG.ORG - MPEG Home http://www.mpeg.org .

[7] History of MPEG http://www2.sims.berkeley.edu/courses/is224/s99/GroupG/report1.html

[8] R. Koenen. Overview of the MPEG-4 Standard. http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm .

[9] D. Marpe and T. Wiegand. G. J. Sullivan. The H.264/MPEG4 Advanced Video Coding Standard and its Applications. Standards Report, Heinrich Hertz Institute (HHI),Microsoft Corporation,August 2006.

[10] L. Matos. Estudo e aplicações da norma de codificação de vídeo H.264/AVC. Master's thesis, Universidade de Aveiro, 2009.

[11] I. E. G. Richardson. H.264 and MPEG-4 Video Compression. Wiley, 1st Edition, 2003.

[12] A. J. R. Neves. Lossless compression of images with specific characteristics. PhD thesis, Universidade de Aveiro, 2007.

[13] JPEG Committee home page. JPEG, 2004. http://www.jpeg.org/index.html.

[14] M. J. Weinberger, G. Seroussi and G. Sapiro. LOCO-I: A low complexity, context-based, lossless image compression algorithm. In Proc. of the Data Compression Conf., DCC-96, pages 140-149, Snowbird, Utah, March 1996.

[15] ISO/IEC. Information technology - Lossless and near-lossless compression of continuous tone still images. ISO/IEC 14495-1 and ITU Recommendation T.87, 1999.

[16] M. J. Weinberger, G. Seroussi and G. Sapiro. The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. IEEE Trans. on Image Processing, 9(8):1309-1324, August 2000.

[17] ISO/IEC. Information technology - JPEG 2000 image coding system. ISO/IEC International Standard 15444-1, ITU-T Recommendation T.800, 2000.

[18] ISO/IEC. *JBIG2 bi-level image compression standard.* International Standard ISO/IEC, 14492 and ITU-T Recommendation T.88, 2000.

[19] Portable Network Graphics (PNG) Specification (Second Edition). Information technology, Computer graphics and image processing - Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E), November 2003.

[20] A. C. Bovik Handbook of Image & Video Processing. Second Edition.

[21] I. Richardson H.264 and MPEG-4 video compression - Video Coding for Next-generation Multimedia.

[22] FFV1, Wikipedia Article. http://en.wikipedia.org/wiki/ffv1.

[23] R. Togni Description of the HuffYUV (HFYU) Codec. http://multimedia.cx/huffyuv.txt

[24] Lagarith home page. http://lags.leetcode.net/codec.html

[25] D. Clark. The popularity algorithm Dr. Dobb's Journal, pp. 121-128, July 1995.

[26] S. Segenchuk. An Overview of Color Quantization Techniques. http://web.cs.wpi.edu/ matt/courses/cs563/talks/color_quant/CQindex.html .

[27] P. Heckbert. Color image quantization for frame buffer display. Computer Graphics Lab. New York Institute of Technology.

[28] A. Kruger. Median-Cut Color Quantization: Fitting true-color images onto VGA displays. Dr. Dobb's Journal, September 1994.

[29] D. Clark. Color Quantization using Octrees: Mapping 24-bit images to 8-bit palettes. Dr. Dobb's Journal, January 1st, 1996.

[30] G. Sharma. Digital Color Imaging Handbook. CRC Press.

[31] O. Verevka, J. W. Buchanan. Local K-means Algorithm for Colour Image Quantization. Department of Computing Science, University of Alberta.

[32] S. W. Thomas. Efficient Inverse Color Map Computation. *Graphic Gems II*, pp. 116-125. Cambridge, Academic Press Professional (1991).

[33] L. Brun C. Secroun. A fast algorithm for inverse color map computation.

[34] Y. L. Ohta, T. Kanade and T. Sakai. Color information for region segmentation. Computer Vision, Graphics and Image Processing, 13, pp. 222-241 (1980).

[35] A. J. Pinho and A. R. Neves. A Survey on Palette Reordering Methods for Improving the Compression of Color-Indexed Images. IEEE Transactions Image Processing, Vol. 13, Nr. 11. November 2004

[36] L. M. Po and W. T. Tan. Block address predictive color quantization image compression. *Electron. Lett.*, vol. 30, no. 2, pp. 120-121. January 1994.

[37] N. D. Memon, A. Venkateswaran. On ordering color maps for lossless predictive coding. IEEE Trans. on Image Processing (5), 1996, pp. 1522-1527.

[38] W. Zeng, J. Li and S. Lei. an Efficient color re-indexing scheme for palette-based compression. *Proc. 7th IEEE Int. Conf. Image Processing.* Vancouver, Canada, September, 2000, pp 476-479.

[39] A. J. Pinho and A. R. Neves. A Note on Zeng's Technique for Color Reindexing of Palette-Based Images, 2004

[40] I. Pinheiro. Automatic Calibration of the Cambada Team Vision System. Master's thesis, Universidade de Aveiro, 2008.

[41] O. Cosma. Image Dithering Based on the Wavelet Transform. Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics. Thessaloniky, Greece, 2004.

[42] Bayer filter, Wikipedia article. http://en.wikipedia.org/Bayer_filter.

[43] http://www.ece.uvic.ca/ mdadams/jasper/

[44] http://iphome.hhi.de/suehring/tml/